Azure

- The 2nd largest public cloud
- The highest growth rate
- It is deployed in most regions
- Most popular in enterprises.

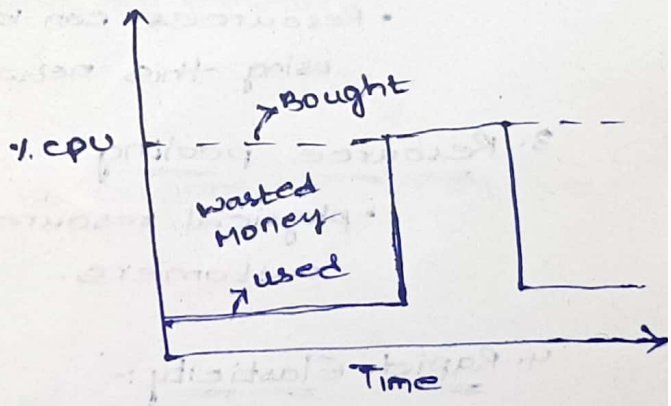
→ Before the cloud

If you need a server, you had to :-

Buy it
install it
Maintain it
Replace it
Have an IT Team

The same goes with :-

Networking
Databases
User Managment ---and more



Cloud :-

Compute, Networking, storage and other services Managed by someone else.

↳ cloud providers :-

- companies who build huge data centers
- fill it with servers, networking, cooling, electricity etc ---
- Design and install various services
- Make it publicly accessible

↳ cloud services :-

- Clouds are huge and the competetion is more
- offer a lot of services :-

AI
IOT
Kubernates --- more

→ In cloud era:-
- create it in the cloud within minutes
- use it as you wish
- pay for what you use
- shut it down when not nedded
- Automatically maintained, patched, secured, monitored.

⇒ Characteristics of cloud Computing:-

1. on-demand self service:-
   - No human interaction is nedded for resource provisioning.
   - Resource can be ~~provided~~ provisioned (created) with a click of a button.
   - provisioning is available 24/7.

2. Broad Network Access:-
   - Resources can be accessed from anywhere using this network

3. Resource pooling:-
   - physical resources are shared between customers.

4. Rapid Elasticity:-
   - resource can be scaled up and down as needed, automatically.

5. Measured Service:-
   - payment is done only for resources actually used.

⇒ CapEx Vs OpEx :- (Capital Expense & operating Expense)

CapEx	OpEx
• Making upfront investment for its future use/profit	• pay for what you actually use

↳ Traditional IT → capEx oriented

- Non optimal
- Not flexible

- extremely flexible
- Most optimal

This is what we get from the cloud.

⇒ Types of Cloud Services:-

1. Iaas:-
   - Infrastructure as a service
   - The cloud provide the underlying platform
     - compute
     - Networking
     - Storage
   - The client handles, and is responsible for all the rest.

2. Ex:-
   Virtual Machines

2. Paas:-
   - platform as a service
   - The cloud provides platform for running apps
   - Including: compute, networking, storage, runtime environment, scaling, redundancy, security, updates, patching, maintenance etc
   - The client just needs to bring the code to run.

Ex:-
Web Apps

3. SaaS:-

Software as a service

- A software running completely in the cloud.
- The user doesn't need to install anything on-premises or in his machine.
- The provider of the software takes care of updates, patches, redundancy, scalability etc.

ex:-

office 365

salesforce

↳ Additional service Types:-

1. faas - function as a service
2. DBaas - database as a service
3. Daas - Desktop as a service
4. IoTaas - IoT as a service
5. Alaas - AI as a service.

⇒ Types of cloud:-

1. public cloud:-

- The cloud is set up in the public network
- Managed by large companies
- Accessible through the internet
- Available to all clients and users
- clients have no access to the underlying infrastructure.

ex:-

Aws, Azure, google cloud, IBM cloud....

2. private cloud:-

- A cloud set up in an organization's premises
- Managed by the organisation IT team
- accessible only in the organisation network
- available to users from the organisations
- uses private cloud infrastructure & engines.

Ex:-
VMWare cloud,
Red hat open shift container platform,
Azure stack.

3. Hybrid cloud:-

- A cloud setup in an organisation premises, but also connected with public cloud.
- workload can be seperated between the two clouds.
- i.e; sensitive data in the organisation's premises public data in the public cloud.
- usually managed by the public cloud, but not always.

Ex:-
Azure Arc,
AWS Outposts

⇒ Azure :-

- Microsoft's public cloud
- Announced in october 2008
- Released in febuary 2010
- The 2nd largest public cloud.
- first focused on paas services
  ↳ to counter AWS's Iaas focus
- Later added Iaas
- currently offers the largest variety of cloud services.

⇒ Regions and Zones :-

- Microsoft build a lot of datacenters for Azure.
- Each datacenter's location is called "Region".
- There are ~60 Azure Regions
- Almost every new resource in the cloud should be allocated to a region.

Zones :-

- Some of the regions have more than one physical datacenter.
- Great for availability in case one datacenter fails.
- Each datacenter is called Zone.
- When there are more than one datacenter in a region, the region is said to be "Availability Zones".
- Some cloud services benefit from availability Zones.

Paired Regions :-

- Some regions have designated pair region.
- for increased availability.
- when full region fails, the other one can fill its place.
- Relevant for some of the cloud services.
- pairs are set by Azure and cannot be changed.

⇒ Azure services:-

Everything that can be done in the cloud is called "cloud service".

Azure having lot of them.

⇒ Account and subscription:-

- subscription is a logical container while account is an identity.
- subscription contains various resources you provision in the cloud (VMs, DBs, networks - - -etc)
- So, every resource is created in a subscription.
- Account is an identity with access to resources in the subscription.

Note:-

A single subscription can be attached to a lot of accounts.

An account can be attached to a lot of subscription

⇒ Azure Basic Concepts:-

Regions:-

- Almost every resource in Azure should be placed in a region.
- How to select regions?
  - ↳ Geographical proximity to system's audience.
    ie, we need to select the region that is closest to the targeted audience.
  - ↳ services availability
  - ↳ Availability Zones
  - ↳ pricing

Resource Groups :-

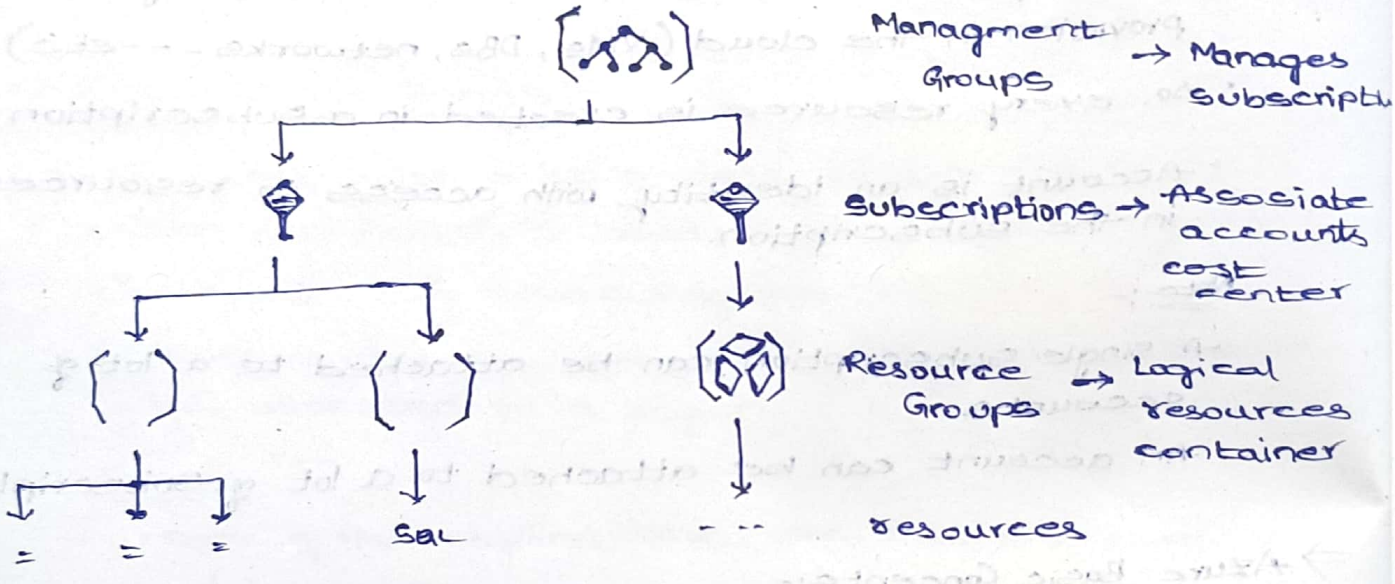
- A logic container for resources
- used for grouping resources by a logical boundary
- free

Ex :-

Development / Test / production resources

Team A resources

Resource Groups vs Subscriptions :-



- Resource groups have naming convension.
  - ↳ "rg" (or) "RG" as part of resource group name.

    This could be prefix (or) sufix

    ex:-

    RG-project-Dev

    project-Dev-RG

- Almost every resource in Azure is placed in a resource group.

Storage Accounts:-

- used to store almost everything.
- used transperently by various services
- used for explict data storage.
- Quite cheap

SLA:- (Service Level Agreement)

- It defines the up time % of cloud service.

ex:-

SLA %	Yearly Downtime Allowed
95	18d 6h 17m 27s
99	3d 15h 39m 29s
99.9	8h 45m 56s
99.99	52m 35s

- So, when creating a new cloud service, always check the SLA of the service.

SLA calculation:-

To get the actual system SLA, multiply the SLA's of the participating services.

ex:- we have;

App service SLA = 99.95%.

Azure SQL SLA = 99.99 %.

Actual SLA = 99.95 x 99.99

= 99.94%.

= 5h 15m 34s annual Downtime.

Cost:-

- Almost everything in the cloud costs money.
- few pricing Models :-
  - ↳ per resource (ie; VM)
  - ↳ per consumption (ie; function App)

for this we are not paying for the creation of the function app what we do pay is the using of the func App.

↳ Reservation (paying upfront)

- Always check resource's cost before provisioning.
- Check for most cost effective alternatives.
- Look for reservations when available and relavent
- we can use "pricing calculator (Azure)" to check for the pricing for the resources.

⇒ Setting Budget :-

cost Managment → Budgets

So, with Budget we can define a budget that we want not to exceed from in our Azure usage.

and when we exceed it (or) come close to exceed it, then we can get alert from Azure.

Click on "Add" then define budget

ex:-

Name : close-to-200

Reset period : Annually

creation date : - - -

Expiration date : - - -

Budget Amount :

Amount : 200

Alert conditions

Type : Actual

% of budget : 90

Action :

Click on "Create"

→ Architects and the cloud :-

cloud-based systems require;

- Infrastructure knowledge
- security
- Hands-on

⇒ Building an App and put it in the cloud :-

App Name : Readit!

↳ It is your bookshop in the cloud.

↳ It contains 4 main services.

Books catalog.

Shooping cart

Inventory Managment

order Engine.

Technical Details :-

↳ developed using .Net core & Nodejs

↳ uses Vs code

↳ uses various databases

→ What we will do ?

- Begin with the basic app, run it locally.
- Move it to the cloud, demonstrating various deployment (compute) options.
- Add networking & security.
- Add database support
- Add messaging b/w various services.
- discuss various alternatives for deployment etc
- keep an eye on the cost.

⇒ Azure Compute :-

- compute is a set of cloud services for hosting and running applications.
- It allows you to upload your code and then run it and it offers various levels of control and flexibility on your application, on your code and the underlying infrastructu on which it runs.

We cover 4 types of compute services:

1. Virtual Machines
2. App services
3. AKS
4. Azure functions

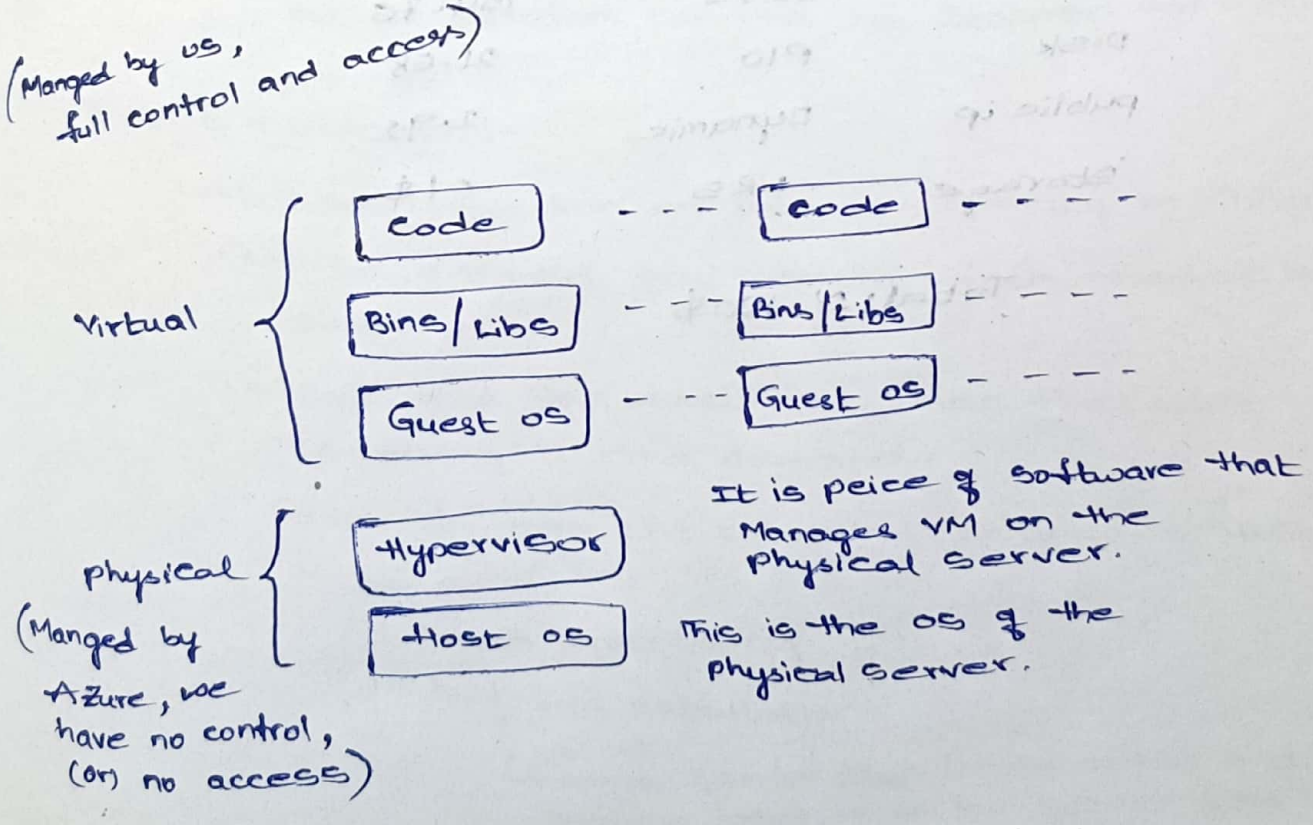
** Virtual Machines :-

A Virtual (Non real) server running on a physical (re server.

- Allows creating new servers extremely quick.
- It is "unManaged service".

(Iaas)

Architecture :-



→ Steps for creating VM in Azure :-

1. Select the location
2. Select the image (os + pre-installed software)
3. Select the size

→ The real cost of VM:-

1. VM:-
   Actual VM
2. Disk:-
3. IP:-
   the public ip exposed by VM
   the price of ip is depend on the exact type of the ip that is exposed, and not all ip address (or) public ip address have a cost.
4. Storage:-
   where the image of VM is stored.
   extremelly low

ex:-

resource	Type	Monthly cost ($)
VM	D2V3	154.76
Disk	P10	21.68
public ip	Dynamic	2.92
storage	LRS	<1 $

Total: ≈ 180 $

⇒ Reducing the cost of VM:-

Most effective techniques to create costs of VM:-

1. Autoshutdown:-

automatically shutdown the machine when not nedded.

↳ relavent mainly for test/dev Machines, which shouldn't be up & running 24/7 but we need them only for specific hours per day.

- By this, we can save >50% of VM cost.

Note:-
when creating VM, we need to select the Auto-shutdown option for enabling.

2. Reserved instances:-

- Allow upfront payment with some discount.
- usually offered for 1 (or) 3 years.
- Great for "production Machine" which runs continiously which runs 24/7.
- offers discount upto 62%.
- can be divided to monthly payments

Note:-
The reservation can not be stopped (or) refunded.

3. Spot instances:-

- Machines that run on unused capacity in Azure.
- can be evicted any moment when needed by Azure.
- offers upto 90% discount, price fluctuates according to the demand.
- Great for non-critical, non-continuous tasks.

ie;

Batch processing

long run calculations

↳ which can be stop in the middle and continue later when the server goes up again.

Note:-
you can enable the spot instance and then you select the eviction type.

4. Disk optimization:-

- make sure to select the right disk for the machine.
- default is premium SSD - the most expensive option.
  ↳ this is the fastest disk and also offers a better SLA.

Note:-

- Select the right size for your machine
- Select Linux over windows when possible
- Check price in nearby regions.

⇒ Availability of VM:-

Mainly four concepts:-

1. Fault Domain:-
   - logical group of physical hardware that share a common power source and network switch.
   - Similar to "rack" in a traditional data center.
     ↳ If there's a problem in power (or) networking in the domain (=rack) → all servers in it shutdown
     ⇓
     You want to make sure your servers are spread across more than one "fault domain (=rack)".

2. Update Domain:-
   - logical group of physical hardware that can undergo maintainance and be rebooted at the same time.
   - Maintanance is done by Azure

If all your servers are in the same update domain - they'll reboot at the same time during maintanence.

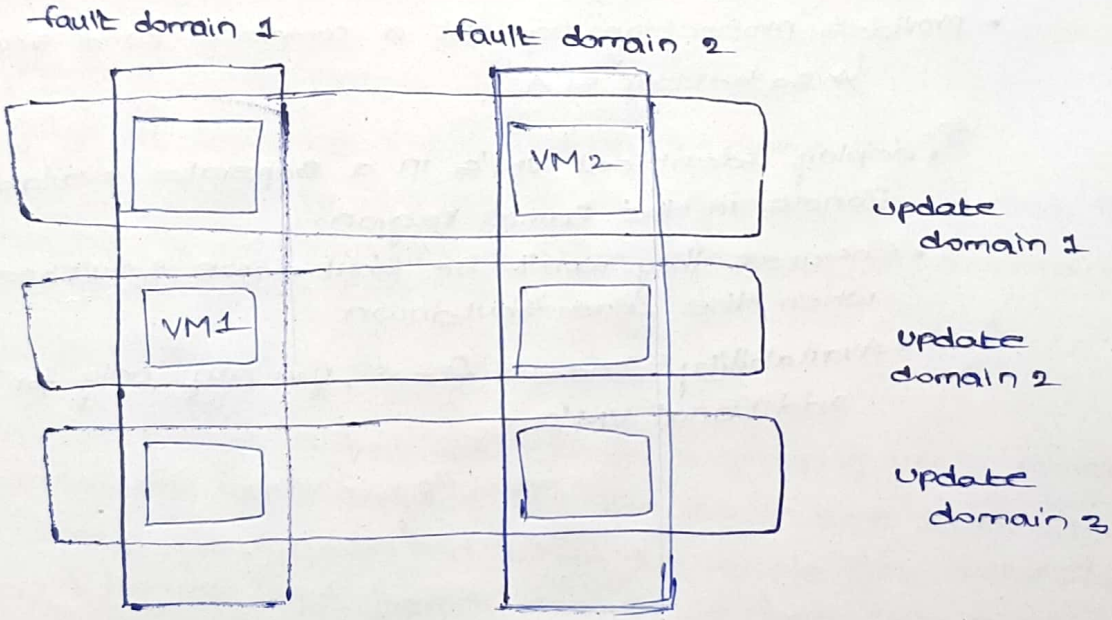
⇓

You want to make sure your servers are spreed across more than one update domain.

3. Availability set :-

- A collection of fault domain and update domain your VM's will be spread across.
- can contain upto 3 fault domains and upto 20 update domains.
- All domains (fault & update) are in same zone (data center) → A physical

ex :-



Let's say, the 2nd update domain is reboot, we still have VM in update domain 1.

Note :-

if we didn't have any availability set the Azure could have decided to place both the machines on the same fault domain because Azure had no idea that these two machines belong to the same application.

So,

- Deploy the identitical VM's into same availability set.
- Ensures they won't be shut down simultaneously when a single fault domain shutdown (or) an update domain reboots.
- Availability set is free, you pay only for the additional VM's.

4. Availability Zone :-

- A physically seperate zone within a Azure region.
- Each zone functions as a fault and update domain.
- Provides protection against a complete zone shutdown.
  ↳ so better "SLA".

So,

- deploy identical VM's in a seperate availability zones in the same region.
- Ensures they won't be shut down simultaneously when the zone shutdown
- Availability zone is free, you pay only for the additional VM's.

⇒ ARM Template :-

Azure Resource Manager Template.

- It is a "json file" describing the resources to be created.
- This file is used by Azure in almost all deployments.
  - ↳ ie; when you create a new resource using the portal, then at the final step of the creation, Azure creates ARM template and then passes it on to the deployment engine of Azure.
    (for almost all the resources this is the process)
- So, ARM template can be exported, modified, uploaded, deployed, because it is basically a file.
- It can be created from scratch.
- It is a "declarative" way of deploying resources.

⇒ Virtual Machine Scale Set :-

- A group of seperate VM's sharing the same image.
- And also VM's are managed as a group, means that they can be scaled out (or) in either manually (or) according to predefined conditions.
- Great for handling unpredictable load.

So, what it means is that we create a group of VM's which are identical VM's and we tell the scale set to monitor these machines and when the scale set detects a heavy load then it can scale them out meaning it can add additional VM's based on the same image and vice-versa.

Note :-

once VM scale set is setup, the machines should not be modified.

ie; should not change files, install apps ---etc.

because new machines created by the scale set will be based on the original image.

- Scale set is free, you pay for the VM's deployed in it.

→ Setting up the catalog App :-

- We need to prepare a "publish version" of the catalog app, means that .NET will create a spatial folder with the files that should be published to the VM. because we can't copy the files to the VM.

  ↳ Steps :-

  Terminal → dotnet publish -o publish

  then, the folder will be created with the files that are required for publishing.

- Create a VM to deploy the catalog App.

  ex :-

  readit-app-rg
  catalog-VM
  Availability Zone
  Zone 1
  Trusted launch VM
  Windows Server 2022 Datacenter
  Size : B2ms
  disk : Standard SSD
  Networking :-
  readit-app-vnet (virtual network)
  Public ip : catalog-VM-ip
  ↳ assigment : Static

- Now, we need to configure this machine as a web server, which is by default.

  So, Server Manager → Add roles & features
  Next → next → select our machine (catalog-VM)
  → next → Select "web server (IIS) role →
  next → next → Role services → next
  Select ;
  custom logging,
  logging tools
  → and "install".

- Now check, wheather our machine is configured as web server (or) not.

  ie; open Microsoft edge
  search "Localhost".

- Before we can install and run our catalog app, we nee to install is "hosting bundle of .Net 6".
  ↳ this allows our machine to host and run .Net 6 apps.

So,
https://dotnet.microsoft.com/en-us/download/dotne
↳ windows → Hosting Bundle

Note:-
Not to download and install the runtime or the SDK, what we need is "Hosting bundle" for running the app in our web server.

So, download and install the hosting bundle.

→ Then, we need to prepare a folder for our catalog app.

So,
file explorer → c drive → New folder (catalog)

Now, copy all the files from the publish folder from the Local machine to this "catalog" folder.

→ Now the last step is to configure the web server on this machine, which is "IIS" in this case since it is windows machine, so we need to configure IIS.

then the catalog folder will be basically a web app.

search bar → IIS → create new website

Sites → Add website

So, we now have our catalog app configured as a web app in IIS.

Note:-

Initially we can't access this machine from the outside. Azure, by default blocks access to VM unless we explicitly allow such an access.

we now only have "RDP" access.

Latter, in networking section we will see how to configure the machine so that the external access is allowed.

⇒ Setting up the wheather Api :-

So, now we need to create another VM (Linux)

Step 1:-

create VM

ie;

reddit-app-rg
weather-VM
image : Ubuntu Server 22.04 LTS
size : B1S
ip : static

Note:-

make sure both the VM's are is same "vnet". since these two machines are going to connect to each other.

Step 2:-

configure this machine to run .Net apps and then install the wheather Api

So, run the commands in the CLI

⇒ Azure Architecture Diagram :-

- when designing an architecture for Azure apps it's a good idea to use Azure symbols in the diagram.
- download Azure icons.

  docs.microsoft.com/en-us/azure/architecture/icons/

Note :-

Never Leave a "VM" open to the internet directly ie; by the public ip address.

** ⇒ App Services :-

- A fully managed web hosting for websites, it allows you to publish your code and then it just runs.

  So, you don't have to create a VM and install anything on it and responsible.

  → Now, because you are not creating the VM then you don't have access to the underlying servers.

- one of the advantage of the App service is that it ** always secured and complaint. (Microsoft is responsible)
- It integrates with many source controls and Devops Engines:

  GitHub
  BitBucket
  Azure Devops
  DockerHub and more ---

So, for example, you can deploy a new version of your app by uploading a new version to GitHub and then App service will automatically pull it and deploy it on your service.

→ Supports following platforms :-

- .Net
- .Net core
- Node.JS
- Java
- Python
- PHP

- Supports "containers".

→ App services supports following App Types :-

- web apps
- web API's
- web jobs (batch processes)

Extremely easy to Deploy :-

1. Develop your app
2. create web app (can be even done by IDE)
3. publish your code.

⇒ App services Tiers :-

when you creating a new app service you will be asked to select "App service tiers".

Note :-

Linux app services are much cheaper than the windows app services.

auto scaling :-

App service can be autoscaled to support spikes and idoad ie; that you can start with the single instance and when the load of the website will suddenly increased then autoscalling feature can automatically add instances to the App service. And after that when everything goes to normal then these instances will be removed.

⇒ Setting the inventory App Service :-

1. Create App Service

↳ create web app

ie; readit-app-rg

name : readit-inventory (it should be unique)

publish : code

Runtime stack : .Net 6 (LTS)

OS : windows

Region : East US

Pricing plan : Standard S1

Note :-

In the App service terminology we basically have two components.

1. App service → it is the peice of code that we are going to deploy.

2. App service plan → it is the underlying infrastructure that will run this code.

2. Deploy your to code to web App from the Vs code.

⇒ Deployment Slots :-

- when uploading code to App service the new version is accessible immediately.
- Sometimes we want to test the version before publishing it so, for that we have deployment slots.
- Deployment slots allows us to upload code and test it seperately from the main site and after validation we swap the slots and promote it to production.
- New slots are created from the portal
- Number of allowed slots depends on plan.

ie; standard plan allows us upto 5 slots.

- Slot is a fully functional app service with a dedicated URL.

- Slots can be accessed directly through their URL
- Slots are free.

Note:-

one of the most interesting features of deployment slots is "traffic splitting".

ie; traffic can be split between slots, which means some users will be routed to the production site (or) the production slot and some to the new slot.

ex:-

you want to deploy the code gradually so that at first only a small fraction of the users will access the new website and according to their feedback you will know wheather you can make new version accessible to all the users.

⇒ Using Deployment Slots:-

Deployment Slots → Add Slot (Staging)

then we will get the new slot with new URL

here we can again deploy the new version code.

→ can split the traffic by giving the traffic %

→ can be swap the slots.

***

⇒ Deployment Types:-

Traffic splitting enables various types of deployment.

1. Basic:-

- All instances update to the new version at once

So, if we have three instances running V1, then in a single click all these instances are now migrated to the new version.

Pros:-	Cons:-
Simple	Risky
fast	System might be unstable

→ Now, how do we implement basic deployment in App Service?

↳ we simply deploy the new version to the App Service and we are not going to use any deployment slots. And in this case, the new version will immediately become the production version of the App Service.

2. Rolling Deployment :-

- Instances are updated gradually in batches.
- only if no errors are found then the deployment resumes.

Pros:-

Allows rollback

Cons:-

Need to support two versions simultaneously.

→ Implementing ?

↳ Deploy to slot then set traffic %, so that the small % will be routed to the new version and gradually increase the % of traffic to the new version untill it reaches 100% and then after yo are sure that the new version is bug free, swap the slots.

3. Blue - Green Deployment :-

- New version uploaded and accesible only to testers.
- After the verification is complete, traffic is routed to new version.

Pros:-

Simple

New environment is always tested

Cons:-

cost

↳ because of more instances

→ Implement ?

↳ first we deploy to a new slot, then the testers will work on the new slot using the url and from the general users nothing has changed.
And only after the testing is complete, we will swap the slots.

Note :-

With App service, while you can stop using stop button, all it will do is to stop the functionality of the App service, but you will still pay for it.

With VM → you pay only when the VM is on

with App service → The only way to stop paying for it is to completely delete it.

***

Azure kubernetes Services :- (AKS)

- It is a managed kubernetes on Azure.
- using AKS, you can deploy containers and manage them using kubernetes tool on Azure.
- There is no payment for the AKS itself, you pay only for the instances (=VM's) that are used by the AKS.

ie; AKS runs on VM's

ie; you don't have access to VM's
you just select the size that you want.

⇒ Containers :-

↳ In traditional deployment :-

- code was copied and build on the production server.
- problems were found on the servers that weren't found in the dev machine.

for this problem, containers appeared.

- containers are a "thin packaging Model".
- They package together Software, its dependencies, and configuration files and then be copied between machines
- This package creates an "automic unit" than can be excecuted using whatever Software and files are contained within it and completely independent from the rest of the machine it's hosted in with one notable exception.
- container uses the underlying operating system.

→ Why containers?

1. predictibility → The same package is deployed from the dev machine to the test to production.

2. performance → containers loads much fast (in secs) than VM (in mins)

3. Density → one server can run thousands of containers but this very same server will run only dozen of VM's.

Note:-
containers are very less lighter than VM's and requiring much less memory.

→ Why not containers?

1. Isolation → containers share the same OS, so isolation is lighter than VM.

Note:-
It's much easier to cross the boundaries b/w the containers than b/w VM's.

If your application contains Sensitive code (or) data it should be probably deployed on an isolated Virtual machine.

⇒ Docker :-

The most popular container environment.

Released in 2013.

Architecture :-

→ "docker daemon" that manages containers (docker server) all the containers running on the host.

Heart of the docker environment

→ docker image using which you create docker container instances.

image of the containers, the containing software to run.

→ docker registory is to store the docker images.

→ clients is to build and manage these containers

Note :-

containers registories can be public.

→ A container is an image that is built and run, the container runs in a sandbox created and managed by the docker daemon.
It is an instance of an image.

→ The client, which is a CLI is where you can send the instructions to the daemon.
This is the gateway for the whole functionality of Docker.
So, using it you can pull images, customise them, runs the containers, list them, shut them off, check the status -- etc

dockerfile :-

- It contains instructions for building custom images.

It can instruct the docker daemon to copy files, run commands, change working directory and more.

Dockerfiles are usually quite small because there are so many baseline images to begin from and the customization required is not that big.

⇒ Support for Docker :-

- Supported by all major operating systems (windows, linux, osx)
- Supported by major cloud providers.

ex:-

amazon ECR

Azure ACR

⇒ Container Managment :-

If there are too many containers we need to manage them.

Major problems are :-

Deployment
Scalability
Monitoring
Routing
High Availability.

So, there are most popular contair managment platforms

⇒ Kubernetes :-

- The most popular container managment platform.
- Released by google in 2024.
- Provides all aspects of container Managment.

ie,

Routing
Scaling
High-availability
Automated deployment
configuration Managment

***

⇒ Azure functions :-

These are small, focused functions runing as a result of an event.

- Great for event driven systems, when you must do something as a response to something else that is happened in your system.
- These functions are automatically managed by Azure.
  ie; Azure can start, stop, scale
- Have flexible pricing plans
- They are serverless.

Serverless :-

It is a cloud resource that is completely managed by the cloud.

- users do not need to think about :-
  VM's
  CPU
  Memory ---etc

Triggers and Bindings :-

Triggers

- The event that made the function run
- There are quite a few triggers
- These are deeply integrated into other Azure services
- Techically not mandatory ie; you can create Azure func which does not have any trigger.

Bindings

- Declarative connection to other resources.
  ie; you can setup connection b/w Azure function and other resource in the cloud.
- So, you can read (or) write value to these resources whether as input, output (or) both.
- These bindings are provided as parameter to the function.
- Makes connection to the other resources extremely easy.
- Not mandatory.

ex:-

1. Run every 5mins (Timer trigger) and calculate the sum of a column in a DB. If it is above 115, send an event in eventGrid (Binding).

2. when a message arrives in the order queue (queue trigger) save it in cosmosDB (Binding) for future handling.

3. Receive Http request (HTTP trigger) with 4 numbers and return the smallest on of them (no binding)

→ Supported Languages:-

C#, Javascript (node.js), Java, Python, Powershell, F#

Cold Start:-

⇒ Azure functions Hosting plan:-

1. Consumption plan:-

pay only for what you use.

calculation ex:-

Assume;

Excecution/month : 9m

Avg. memory consumed/excecution : 800MB

Avg. excecution duration : 1.5sec

⇓

Total seconds : 9m * 1.5sec

Total GB/sec = 13.5m * 10.8m - 400k free grant

= 10.4m GB/sec

payment for excecution time : 10.4m * 0.000016$

= 166.4$

payment for excecution : 9m - 1m free grant

= 8m * 0.2$/m

= 1.6$

Total payment = 168$

2. Premium plan:-

pay for pre-warmed instances (hosts)

pay for scale out instances.

No cold starts

No memory limit (upto host RAM)

Better performance

VNet integration

predictable price

3. Dedicated plan:-

The functions run on an existing App service.

⇒ Durable functions:-

These are stateful functions that interact with external resources and keep track of flow.

- offers very simple syntax.
- These are very useful for function chaining.
  ↳ calls various functions sequentia and apply the output of each function to the next one.

Note:-
We need to install "Azure functions core Tools" and these allows us to run the azure function locally on our machine.

Link:- lear.microsoft.com/en-us/azure/azure-functions/functions-run-local

- Azure functions are basically "Api's".

Handson:-

Create function app in Azure
Then deploy the functions from local
Then test them.

FN app:-
rg: ---
name: harishorderfnc
runtime: .Net6
version: 6 (LTS)
os: windows
hosting plan: consumption plan

deploy code:-
Goto vs code
↳ Azure extension
↳ select the fnc which is created from the subscription.
↳ In the workspace
↳ click on "func" icon
↳ "deploy to function app"

storage:-
create new storage account.
ie;
harishreaidfnc6torage
→ "create"

Note:-
After deploying the code, we get the message
↳ click on "upload settings", so settings will be uploaded.

→ How to choose Compute Type ?

(follow the chart)

→ More compute options:-

1. Logic Apps
2. ACI - Azure container instance
   ↳ which runs docker container without the AKS.
   So, if you don't need all the complexity of kubernates but just want to run a single container instance that is the way to go.
3. App service container
   ↳ allows you to deploy Docker to App service

Networking:-

Networking is the foundation of cloud security.

Major 4-Networking cloud services:-

1. VNet
2. SubNets
3. Load Balancer
4. Application Gateway.

1. VNets (Virtual Networks):-

- A Network in which you can deploy cloud resources
- Many cloud resources are deployed in VNets

  ex:- VM's, App services, DB's - - - etc

- "Virtual" in the Virtual Network means that this network is based on physical network and logically seperated from other Virtual Networks.

  so, when we create a new Virtual network we don't really create a physical network, we use an existing resources that are part of Azure and inside the Azure network we create a logically seperated network.

Note:-

The resources in same VNet can communicate with each other by default, but not with resources in other VNet's. (this can be modified).

- In AWS it's called "VPC" (virtual private cloud)

  ie; other organization's VNets cannot communicate with your VNet.

- VNets are free.
- Limit of 50 VNets per subscription across all regions.

Characteristics:-

- scoped to a single region
  ↳
- scoped to a single subscription.

- VNet's can be connected via "peering".
  - ↳ the resources in one VNet can communicate with resources in other VNet.
- VNet's are segmented using subnets.
  - which is a logical group of resources with in a VNet and is protected using "NSG" which is defined on the subnets not on the VNet.

So,
Major role of VNet is for security.

So, the most important thing to think about when designing network is:

"How to limit access to the resource in the VNet so that risk is minimized".

- Each VNet has its own address range (or) IP range.
  - by default - 65,536 address can be customised.
  - All the network devices must be in this address range.
  - Address range of a Virtual Network is expressed using "CIDR Notation".

⇒ CIDR Notation:-

classless Inter-Domain Rounting.

This is a method for representing an IP range.

This composed of an address in the range and a number b/w 0 and 32.

This number indicate no. of bits allocated to the address.

The small the number the larger the range.

ex1:-

109.186.149.240 ← IP address

↓ ↓ ↓ ↓

8bits 8bits 8bits 8bits

To this ip address, we add the range. ie; 24 bits

ie;

109.186.149.240 /24

24 bits allocated to address

8 bits allocated to range

So, 109.186.149.000 – 109.186.149.255

total = 256 addresses.

ex 2:-

109.186.149.240 /16

16 bits allocated to address

16 bits are allocated to the range.

So, 109.186.000.000 – 109.186.255.255

total = 65,536 addresses

ex 3:-

109.186.149.240 /20

20 bits allocated to address

12 bits allocated for range.

⇒ for calculation of actual range :-

149 decimal = 1001 0101 Binary.

↳ cannot use this.

So, 1001 0000 Binary = 144 decimal

So,

109.186.144.000 – 109.186.159.255

total = 4096 addresses

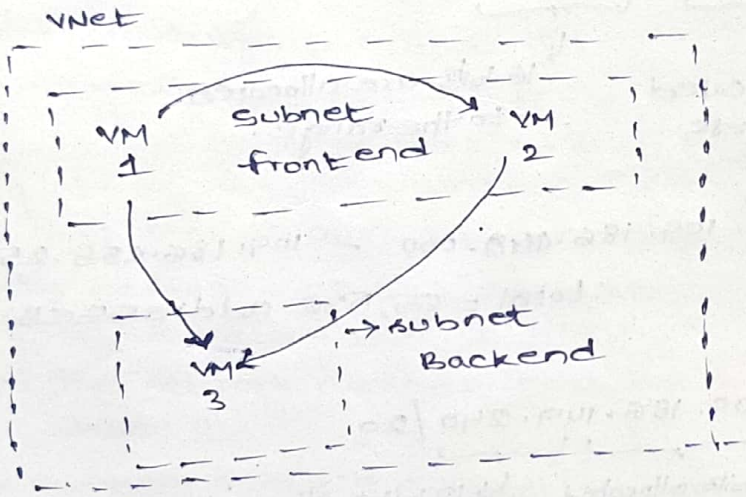
2. Subnets:-

- A logical segment in the Virtual Network
- It shares a subset of the VNet's IP range.
- used as a logical group of resources in the VNet.
- It is must, resources must be placed in a subnet, cannot be placed directly in the VNet.

you will also, when you create VNet, a default subnet is created alongside this VNet. (named default)

- Resources in the subnet can talk to the other resources in other subnet in the same VNet. (By default can be customised)

ie,



- Each subnet gets a share of the parents VNet's IP range.

Note:-

Never use the full range of the VNet in a subnet.

- once we set the address space of the subnet, it is extremly hard to modify it later.

Pricing:-

- Subnets are free
- Limit of 3,000 subnets per VNet

Note:-

Azure need "5" internal ip addresses for own internal managment for each subnet.

Note:-

⇒ When we create two VM's in two seperate VNets their private IP addresses can be same, and that shows why we cannot connect from one VNet to other VNet.

But this can be done using "peering".

⇒ Network Security Group:- (NSG)

It is a gatekeeper for subnets.

It defines who can connect in and out of subnet.

In other words, you can think of it as a mini-firewall

NSG should be a standard part of subnet creation.

So, whenever you create a new subnet makesure there is a NSG attached to it.

NSG is free.

→ How NSG works?

↳ It look at 5 tuples:-

↳ Source → where did the connection come from

↳ Source port → The port the source is using.

↳ Destination → where does this connection request goes

↳ Destination port → To which port does it want to connect.

↳ protocol → TCP, UDP, Both

So, Based on these 5 tuples the connection is either allowed (or) Denied.

This is called "security Rule".

- Each rule is assigned a Number.
- Lower the Number, the higher the priority of the rule.

NSG & VM's:-

An NSG is automatically created and attached to every newly created VM's network interface.

- By default, the NSG of a VM opens the RDP on windows machine (or) SSH on Linux machine to anyone.
- This must be handled first thing after creation.

Network interface :-

In VM, a network interface is a virtual representation of a physical network adapter.

It allows the VM to communicate with other devices on the network, such as VMs, physical servers, (or) external services.

It functions similar to the physical network interface card (NIC), enabling the VM to send and receive data packets over the network.

Note:-

You can connect NSG to either subnet or to a Network interface (or) both.

Because NSG is an independent resource that can be attached to multiple resources.

→ Usually there are two types of Rules:-

1. Inbound port Rules:-

These will dictating which traffic can reach into the VM.

Every rules has priority.

ex:- rules having priority 65000 (or) 65500

2. outbound port rule:-

Note:-

NSG is an essential part of every network device that you use in Azure.

NSG rules effects the external access, so only traffic incoming from the outside will be effected.

→ Network peering:-

Sometimes, to increase security we want to place some resources in a completely different VNet's.

Main reason:-

Not to place non-public resources in a VNet that has public access.

So, Network peering allows two VNet's to connect to each other.

from the user's point of view it is a single VNet.

while defining peering we should make sure address spaces are not overlapped, because if those VNet's share the same address space we can not peer them.

- Use NSG for protection.
- peering can work across regions.
- It is Not free.

Note:- we cannot peer to VNets with overlapping address

ie; outbound data transfer

100 GB × $0.0100 per GB

Inbound data transfer

100 GB × $0.0100 per GB

→ using network peering:-

↳ weather VNet

↳ peerings

↳ Add peering

Note:-

when configure peering we basically configure two peerings, one ~~for~~ from for our VNet to the remote one and the other one for the remote VNet to our VNet.

we configure both peerings from the same page.

⇒ Network Watcher :-

This is basically a suite of tools, that you can use when working with networking.

You can use it to visualize and troubleshoot your networking.

↳ connection troubleshoot :-

This allows you to troubleshoot connection between two resources in Azure.

Note :-

Not all resources are supported here.

Usually you will use it to troubleshoot connection between two VM's.

So, we need to do is; fill the details and then click on "run diagnostic tests" and you will receive if there is any problem with direct connection b/w these two resources, then this tool will show you where exactly the network is stopped.

↳ Topology :-

Here we can see high-Level View of the networks in your subscription.

Hands-on :-

→ Setting up the NSG for catalog VM :-

↳ Add the NSG rule

Source : Any

destination : Any

Service : custom

destination port range : 8080

protocol : Tcp

Priority : 310

Name : Allow8080Inbound

description : Allow access to the catalog app.

Note :-

If still didn't work (ie; NSG rule doesn't take effect)

1. Wait for 2-3 min and try again
2. connect to VM & turn off completely windows defender. (because it might be blocking external traffic).

→ Setting up the NSG for ~~cool~~ wheather Api VM :-

↳ edit SSH rule

Source : My ip address.

↳ Add NSG rule for accessing VM from cloud shell

source : Azurecloud

destination : Any

service : SSH

⇒ Secure VM Access:-

- The larger the attack surface, the greater the risk.
- we want to minimize it as much as possible.
- Leaving public IP is always a risk we want to avoid.

→ What can be done?

we have 4 Techniques

1. JIT Access
2. VPN
3. Jump Box
4. Bastion

1. JIT Access:-

Just IN Time Access

- using JIT access we can open the port for access on demand and the automatically close it.
  So, port is not always open to the internet.
  → ie; when we want to connect to the VM, either using RDP or SSH only then the port is open.
- JIT access can be configured from the VM's page in the portal.
- But it requires security center license upgrade (we don't have for free trail)

Handson:-

Any VM → configuration item

Then you need ↵ to update the license.

2. VPN :-

Virtual private Network

- It is a secure tunnel to the VNet.
- can be configured so that no one else can connect to the VNet.
- for this you required VPN software and license, which are not part of Azure.

3. Jump Box :-

- place another VM in the VNet
- Allow access only to this VNet
- when need to access one of the other VMs, then connect to this one and connect from it to the relevant VM.
- So, only one port is open
- cost only for the additional VM (Jump Box)

4. Bastion :-

It is a web based connection to the VM
No open port is required
Simple and secure
costs money around 140 $/Month

Hands on :-

When connect to the VM
click "connect via Bastion".
↳ Deploy Bastion
↳ Then connect using "username" & "password"

Note :-

Even though there is no NSG rule allowing RDP access to the machine we can still connect through Bastion.

So, Bastion allows us to connect to VM in azure even though their RDP access is blocked.

⇒ Service Endpoint :-

A lot managed services exposes public IP

ie; Azure SQL server,
App Service,
Storage and more---

Sometime these resources are accessed only from resources in the cloud.

ie; Database in the backend.

→ Service endpoint solves the security risk.

It creates a route from the VNet to the managed service

→ How to use?

- Enable service endpoint on the subnet from which you want to access the resource.
- on the resource, set the subnet as the source of traffic.

⇒ Private Link :-

Newer solution for the above problem.

It extends the managed service into the VNet, ie; traffic never leaves the VNet and it stays private all the time.

- Access from the internet can be safely blocked.
- Not free

→ How to do?

- configure the resource to connect to the VNet.

and the we create a private link b/w VM and the SQL and then we need to configure private DNS which will handle the routing b/w the resources.

⇒ Service Endpoint Vs Private Link :-

	Service Endpoint	Private Link
1. security	connects via public IP	connects via private IP
2. simplicity	very simple	More complex
3. price	free	Not free
4. supported devices	Limited List	Large List, probably will get Larger
5. on-pem conectivity	Quite Complex	Supported.

⇒ VNet Integration :-

It allows access the app service to connect into the VNet and access the resources that are placed inside it.

Note:-
VNet integration supports same region VNets.

The service endpoint and private link will allow access from a resource that is located inside a VNet to a managed service.

VNet integration allows connecting from managed service to a resource that is located inside a VNet.

→ When we integrate to VNet, then we have to connect to a subnet that is completely empty, and resources in it.

⇒ App Service Access Restrictions:-

- It is similar to NSG, but for App service.
- It is for "restricting" traffic to the App service.
- By default, all inbound traffic is allowed in the relavent ports which by default are 80 and 443.
- Using access restrictions inbound traffic is restricted to the allowed IPs/VNets/Service Tags.
  - ↳ It is a general name for a managed Azure service such as Load balancer (or) Application Gateway (or) Azure Monitor and so on...
- One of the main use cases for access restriction is when our App service is a backend App service that should be accessed only from front end App service (or) a VM, should not be accessed from the internet.

Hands on:-

open any app service
- ↳ click on "default domain" to open the web page.

open "Networking" tab
- ↳ Go to "Access restrictions"
  - ↳ Here, we can add the rules to restrict the traffic

⇒ ASE:- (App service Environment)

- It is a spatial type of app service, which is deployed directly to a dedicated VNet.
- This VNet can be configured like any other VNet, SubNets, NSG - -etc
- It is created on dedicated Hardware

Note:-

we cannot place the app service inside VNet, because the app service by default is exposed to the world.

- ASE is quite Expensive.

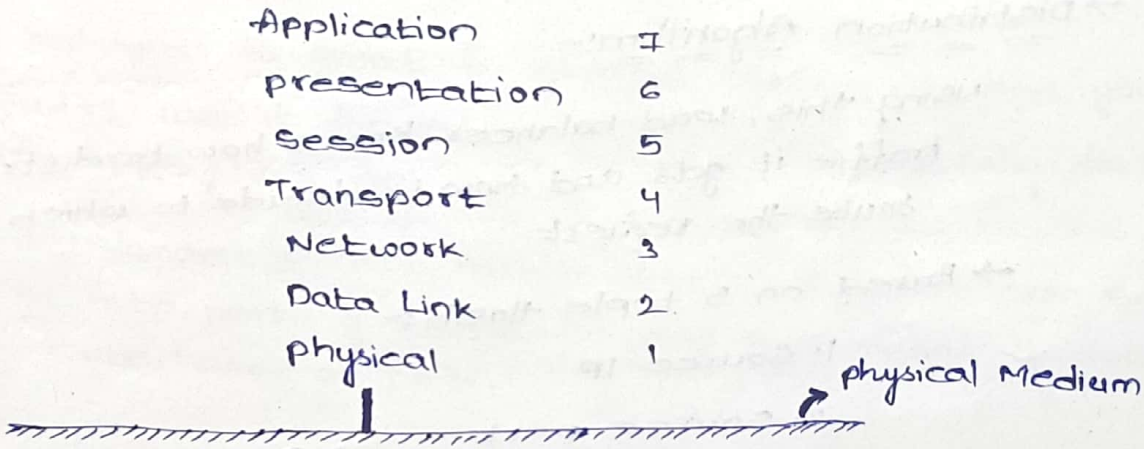
*** Load Balancer :-

Load balancer is an Azure service that distributes load and checks health of the underlying VM's

When VM is not healthy, then no traffic is directed to it.

- Load balancer can work either with VM (or) scale set.
- It can be public (or) private ie; it can either be accessible from the internet or used only within a closed VM.
- It operates at "Layer 4" of the OSI Model

OSI Reference Model :-



→ Communication between computers deals with the OSI communication model.

This model seperates the connection components to several layers which are from top to bottom.

→ Layer 4 of OSI Model is the layer that is familiar with the actual transport in the network so it knows about IP, port, protocols, TLS and more but it doesn't know the actual content of the communication.

It only knows the technical data of the communication.

→ Layer 7 interact with the application, so we can see the content of the transmission.

ex:- http URL + path + params

So, Load balancer is dealying with Layer 4.

→ Typical deployment of Load balancer:-

Ex:-

There are two Load balancers

1. public
2. internal

So, the public load balancer is exposed to the internet via port 80 and it gets the traffic and distributes it accross the VM's and then the traffic from the VM's goes further down to the internal load balancer which distributes the traffic to another set of VM's, this time with port 443.

→ Distribution Algorithm:-

Using this, load balancer knows how to distribute the traffic it gets and how to decide to which VM to route the request.

→ Based on 5 tuple Hash:-

1. Source IP
2. Source port
3. Destination IP
4. Destination port
5. Protocol type

Same tuples used by NSG.

→ Load Balancer Types:-

Basic	Standard
No redundancy	Redundant As, if there is a problem with the load balancer, we have another instance which automatically starts working.
open by default	Secure by default
Upto 300 instances	Upto 1000 instances
No SLA	99.99% SLA
free	Not free

→ Configuring load Balancer :-

4 Main Configurations

1. frontend IP configuration → The public IP exposed by the load balancer
2. Backend pools → The VM's connected to load balancer.
3. Health probes → Probes checking the health of the VM's. Non healthy VM will not be routed to.
4. load balancing rules.
   ↳ A rule connecting frontend IP with Backend pool

Note :-

Load balancer is great for internal resources and you shouldn't use it for external resources espically for the web Apps / web API --etc

- Load balancer cannot handle Http, it doesn't even see the http parts of the traffic and that means it doesn't route based on path.

⇒ Application Gateway :-

It is a web traffic load balancer.
(or) a load balancer with improved web capabilities.

- It can function as an external endpoint of the web app
- It works with —
  - VM
  - VM scale sets
  - App services
  - Kubernetes (requires some Hacking)

- It is the URL-based routing which allows us to route based on the URL path. So that the request contains a specific path the application Gateway will know to route this request to a specific resource.

- It operates at Layer-7 of OSI Model.
  It can see the content of the transmission ie; http URL and the path and parameters.

WAF :-

Web application firewall

It protects web apps against common attacks

ie;

cross-site scripting.

SQL injection ---

- WAF can work in detection or) prevention mode.
  - ↳ with detection mode, when the WAF detects the security related problem in the traffic it simply notify you and warns you but it will not deny the traffic.
  - ↳ with prevention mode, the WAF will actively block the traffic and will also notify you all the way.
- Many organisations have their own WAF deployment.

→ Application Gateway Sku's :-

1. Standard_V2
   - ↳ includes all the feautures excluding WAF
2. WAF_V2
   - ↳ includes everything (double the price)

→ Application Gateway Networking :-

- It should be placed in its own subnet.
- often in its own VNet.
- Must make sure backend resources are :-

  accessible from the AG subnet

  Not accessible from anywhere else ---

→ configuring Application Gateway:-

5 Main Configurations.

1. Backend pools — The VMs, scale sets or App services connected to the Application Gateway.

2. Http settings — Settings for the incoming Http requests

3. frontend IP configurations — The public IP exposed by the AG.

4. Listiners — receives requests on the specific port and protocol.

5. Rules — A rule connecting listiner with backend pools.

Hands on:-

→ Creating Application Gateway:-

Application Gateway

↳ create

Basics:-

Resource group : readit-app-rg

Name : read-app-gw

region : East US

Tier : Standard V2

Enable autoscaling : No

Instance count : 1

Availability Zone : None

HTTP 2 : Enable / disable (Any) ✓

Virtual Network : Create New

↳ Name : readit-appgw-VNet

Note:- make sure that the address space of this VNet does not overlap with the Catalog app VNet because we need to peer these two VNets.

SubNet Name : gw-Subnet

frontends:-

Here we are going to define what exactly the AG is going to use in order to listen to requests coming from outside.

Here, we are going to use this AG to listening external requests.

Therefore;

frontend IP address type : Public

Public IP address : Create New

↳ Name : readit-ip

Sku : Standard

Assignment : Static

Backends:-

Here we are going to define "Backend pools"

Note:- Backend pool in AG is where the AG is going to route the request to.

Here, we are going to create "2" Backend pools.

1. Inventory App
2. Catalog App

Click "Add Backend pool"

↳ for Inventory App Service

Name : inventory-pool

Target type : App-service

Target : readit-inventory-mem1

↳ for catalog App

Name : catalog-pool

Target type : Virtual Machine.

Target : "Here we don't see the catalog VM" because catalog VM is placed in different VNet

So, for now set "Add backend pool without targets" to "yes".

Configurations:-

Here we are going to define the routing rules.

These rules define how exactly the request is going to be routed from the frontend IP that we defined to the backend pools.

Click "Add routing rule"

↳ for catalog App

Rule Name : catalog-rule

Priority : 100

Now, we are going to define the listener and the backend targets.

→ Listener defines how exactly we are going to work with the public IP and the backend targets defines how exactly we are going to connect to the backend of this rule.

↳ Listener:-

Listener name : catalog-listener

frontend IP : public

Protocol : HTTP

Port : 8080

Note:-

Port can be any port

-this port is, what we are going to listen in the frontend-ip

In this case, we cannot use the Https protocol, because we don't have any certificate installed on these application gateways

↳ Backend Targets:-

Target type : Backend pool

Backend target : catalog-pool

Backend settings : Add New

Here, we are going to define how we are going to call the backend in the backend pool.

Name : catalog-settings

protocol : Http

port : 8080

rest : Leave by default

for Inventory App:-

Rule name : Inventory-rule

Priority : 101

↳ listiner:-

Name : Inventory-listener

frontend ip : public

protocol : Http

port : 80

Backend targets:-

Target type : Backend pool

Backend target : inventory-pool

Backend settings: Add New

↳ Name : inventory-settings

protocol : Https

Note:-
Https because this is how App service is expecting us to call it. so even though we are calling the AG using the Http protocol and the 80 port, the AG still is going to use the Https protocol using the 443 port to call the App service.

Backend server
Certificate issued : "Yes"
by

then click "Add".

Tags:-

Leave default

↳ review + create

↳ create

Note:-
will take 10-20 mis for deployment.

→ Connecting Inventory App to Application Gateway:-

After deploying the AG, you might see the error.
This error is that the instances in the backend pools are not Healthy.

Click "Backend Health"

we can see the Health status here.

click "Backend settings"

↳ Inventory-settings.

↳ change "override new host name" to "yes"

Reason:-

The request to backend App service is going to be done using the Https protocol.

So, the domain name used in the request should match the domain name that is set in the SSL certificate.

But, we are calling the AG frontend and the App service front end, then these names do not match.

Therefore, we need to override the name check that is done with a certificate with the name of the App service.

Also;

Set;

Host name override : pick hostname from backend target

i.e; we want to use the host name of the App service as the host name of the front end will use.

"Save" the settings.

Now;

Copy the public ip of "AG"

Paste it in new tab of browser

so, we will access the inventory app service through the AG.

Note:-

default is "80 port".

Note:-

we are also accessible the inventory app through App service. we need to restrict that access.

So, in order to protect the App service, we are going to use service end points.

⇒ service end points is one of the way to secure connection between Azure resources.

So,

Go to AG

↳ click on VNet

↳ click "service end points"

Now we are going to add service end points to the subnet of the Application Gateway.

↳ click "Add"

Service : Microsoft.web (since we are connecting to App service)

Subnets : gw-subnet

Now, Service endpoint is created.

Now, we want to restrict access on the App service side so that only traffic from this VNet will be allowed.

Goto "App service"

↳ Inventory App service

↳ Goto "Networking"

↳ Click "Access restriction"

↳ Add rule

Name : gw-access

Action : Allow

Priority : 100

type : Virtual Network (because we want to allow trafic from the AG VNet)

Virtual Network : readit-appgw-vnet

Subnet : gw-subnet

Note:-

If we wouldn't have defined service endpoint for this subnet, at this stage we would see an error message.

↳ "Add rule".

↳ Save

→ Connecting Catalog VM to Application Gateway:-

first define peering b/w these two VNets.

So;

Goto catalog VM

↳ VNet

↳ peerings

↳ Add peering.

readit-gw-~~readit~~peering

gw-readit-peering.

Virtual Network : readit-appgw-VNet

"Add".

Then;

Configure the backend of Gateway to connect to our catalog VM

So;

Goto catalog VM

↳ copy ip address (private ip)

Goto "AG"

↳ Backend pools

↳ catalog pool

↳ Target type : ip addresses

target : paste the catalog VM private ip.

"save"

Note:-

restrict the access from the "Catalog VM"

ie; delete the rule which is set to access the app from the browser".

ie; Allow8080Inbound.

→ Check wheather we can access the catalog app through AG.

ie; AG public ip : 8080.

⇒ Application Gateway and AKS :-

No built-in integration with AKS

AKS has kind of Gateway (= services)

There is Application Gateway Ingress Controller (AGIC) that does this.

⇒ Application Gateway and function Apps :-

- function apps are basically App services.
- They can be protected by Application Gateway the same way App services are.

  ie; configure them in Backend pool

  Configure Access restriction rules

***

Data In Azure :-

Azure provides many cloud solutions as cloud services.

ie; Relational databases
NoSQL databases
Object stores ---

These are fully managed services.

These can be part of Azure applications (or) completely different.

Major database features :-

→ what to look for when selecting a database;

1. Security — Network isolation
   ↳ ie; No one can access your database if not allowed to.
   Encryption
   ↳ ie; someone steals the disk then he won't be able to access the data.

2. Backup — Backup type
   Rentintion period

3. Availability — SLA, Replication

Database on VM :-

We have the option to install database on VM.
There are readymade VM's that already containing and comes pre-installed with databases.

ex:-
Oracle Database 12.2.0.1 Enterprise edition

Pros:-
full flexibility
full control

Cons:-
You have to take care of everything

⇒ Azure SQL :-

- A managed SQL Server in Azure.
- Works like any other SQL server using the same tools.
- Great compatability with on-prem SQL Server.
  - ↳ depends on the Azure SQL flavor
- offers built-in security, backups, availability and more
- flexible pricing models.

Azure SQL flavors :-

1. Azure SQL database :-
   - It is a single database on a single server.
   - Automatic updates, backups and scalling.
   - Good compatibility with on-prem SQL server.
     - ↳ but not all features are supported.
   - Security
     1. IP firewall rules
     2. Service endpoints
     3. SQL & Azure AD Authentication (Active directory)
     4. Secure communication (TLS)
     5. Data encrypted by default (TDE)
        - ↳ transparent data encryption.
   - Backup
     1. full → every week
     2. differential → Every 12-24 hrs
     3. Transaction Log → Every 5-10 mins
   - Rentention Period
     1. Regular backup : 7-35 days (default-7)
     2. Long term backup : upto 10 years
   - Availability
     - Backup is stored in a geo-redundant storage.
     - SLA - 99.9% - 99.995% depends on teir and redundancy.

- Compute Tiers

There are two :-

1. provisioned :-
   - Pay for allocated resources regardless of actual use
   - can be reserved.
2. serverless :-
   - Pay for actual use
   - Automatically paused when inactive (pay just for storage)
   - Slight delay when warming up
   - can't be reserved.

2. Elastic pool :-
   - Based on Azure SQL
   - Allows storing multiple databases on single server. This is great for databases with low average utilization and infrequent spikes.
   - cost effective

3. Managed Instance :-
   - It is closer to the on-prem SQL server.
   - Near 100% compatability with on-prem SQL
   - Also you can deploy this into VNet.
   - No active geo-replication
   - SLA : 99.99%.
   - Supports built-in functions
   - No Auto scalling & tuning.
   - No availability Zone
   - No serverless tier.
   - No Hyperscale

⇒ Which Azure SQL to choose?

Are you migrating an on-prem SQL?

↳ "Managed Instance".

Do you need multiple, mostly low-utilization DB's?

↳ Elastic pool

All other cases

↳ Azure SQL

⇒ Hands ON :-

Creating & connecting to Azure SQL :-

SQL databases

↳ Create

Basic :- rg : readit-app-rg

db name : readit-db

server : create New

↳ server name : (should be unique)

harishreaditdbserver

Location : East US

Authentication method : use SQL authentication

→ give username & password.

A server is basically the underlying hardware that the database is using.

SQL elastic pool? : No

Workload environment : Development

~~Configure~~ +

compute + storage :

↳ Service tier : Basic

Backup storage redundancy : Geo-redundant backup storage.

Networking:-

Connecting method : No access

Min TLS version : TLS 1.2

→ "Create"

Now, We are going to use "VS code" to perform operations on database.

So: open VS code

install "SQL server extension"

ie; SQL server (mssql)

Click on extension view after installing.

↳ "Add connection"

copy the connection string from the Azure SQL

ie; ADO.Net (SQL Authentication)

Then paste in "VS code" and replace the password you created.

& enter

Then we can set a profile name

ie; readit db

If we get an error.

↳ click "set server firewall"

Public network access : select networks

Add your ip address in "firewall rules".

→ Connecting the catalog to the Database :-

open "catalog app" code in VS code.

we have some changes that we need to do in code, inorder to configure it to work with Azure SQL.

1. Startup cs file :-

Set "useInmemory" = false.

2. Appsettings.json file :-

Place the "connection String" Here.

And, there are some commands that we need to run that will configure the entity framework, which is a mechanism in dotnet for working with database.

So, we are going to configure it to work with Azure SQL.

1. Install migrations tool :

dotnet tool Install --global dotnet ef --version 7.0.14
(compatable with dotnet 6)

2. Create the migration files.

dotnet ef migration add -c BookContext InitialCreate

3. Update the database

dotnet ef database update

Now, you can see New table is added in the database.

↳ GotO "Mssql extension"

↳ db

↳ tables.

Now, run the code in local

you will see the books which are in cache in table.

→ Another way to take a look at data of the database.

Goto portal
open the Azure SQL which is created.
open "Query editor"
Login Here.
↳ Here you can see the tables.

Now, publish the code to the Catalog VM

→ open Terminal;

dotnet publish -o publish

open the catalog app VM
↳ Goto IIS
↳ Goto "catalog" site and "Stop" it.

then; open file explorer.
↳ open catalog folder and replace all the files.

then; Start again

Note:-

Add the "AG" ip to the server firewall in Azure SQL

Securing the database connection:-

Now, the catalog VM connects to the Azure SQL using public IP address. (we add the firewall rule)

But we don't want that.

We want the connection from the catalog VM to the Azure SQL to be protected and go through "private IP".

So, for that we have two mechanisms.

1. service endpoint
2. Private endpoint

We will use "private endpoint" here.

Goto Azure SQL

↳ Networking. (set server firewall)

↳ private access

↳ create private endpoint

ie,

rg : readit-rg

Name : readdb-ep

target sub resource : sqlserver

VNet :-

VNet : readit-app-vnet (where the catalog app is located.)

subnet : default

DNS :- (No changes)

Here DNS zone will redirect calls to the SQL server to the private ip instead of public ip.

Tags :-

No changes

Now, we can completely disable the "public Network access".

But for now, delete the "catalog VM" firewall rule.

→ Connecting Inventory App to Azure SQL :-

open the "inventory app" in VS code.

↳ open index.cshtml.cs

uncomment the lines (ctrl+k+U)

→ deploy to web App

for placing the connection string.

Goto "App service"

↳ configuration

↳ Add connection string.

Name : BooksDB

Value : ----

Type : SQLAzure

Now, the outbound IP address of the App service is not allowed in the database.

We can add the "firewall rule", but we prefer to secure the connection.

→ The way to secure the connection from App service to other serverless resources is through "VNet integration".

So, we have two steps here.

1. we need to have VNet integration from the App service to the VNet where the database is.
2. Also we need to have the private endpoint setup b/w the database and this VNet.

We already have the private endpoint setit up for the catalog'.

So;

Goto Appservice

↳ Networking.

↳ VNet integration

↳ Add virtual Network integration

ie;

VNet : readit-app-VNet

Subnet : wheather-Subnet

(or)

Create a subnet named "VNet-integration-subnet".

Note:-

Here select the VNet which database linked ie; through private link

Note :-

Subnet must be empty for VNet integration.

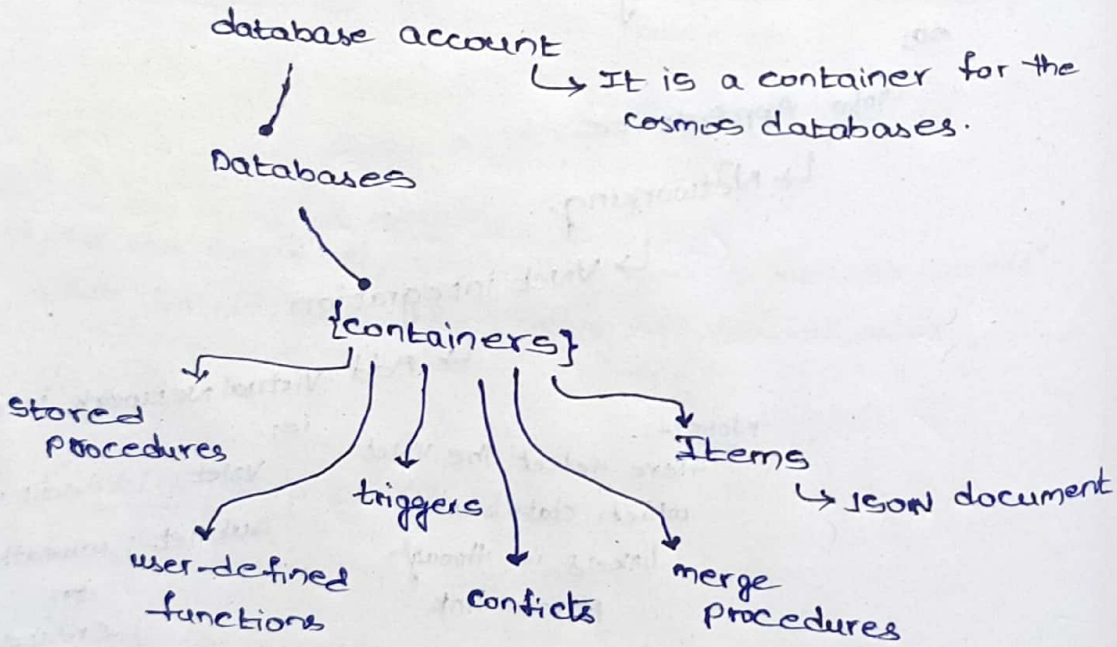
⇒ CosmosDB:-

- fully managed NoSQL database.
- Microsoft commits to "less than 10 milliseconds" for 99% of operations against this database.
- Globally distributed.
- fully automatic managment - updates, scaling -- etc
- It exposes multiple API's

  SQL, mongo, Gremlin, Azure table, Cassandra

  ie; when you create cosmos db you can choose how you want to use this database.
  ie; if you familier with SQL, then you can use "SQL API"

- Hierarchical:



- Availability:-

  can be distributed across many regions.
  API automatically picks the closet one.
  when using write replication SLA is 99.999%

  ↳ highest SLA in Azure

- **Backup:-**

  full backup every 1-24hrs (4 default)

  Retention period 20-30 days (default is 30)

- **Security:-**

  IP firewall rules

  Service endpoints

  Private Endpoints

  Azure AD Authentication

  Secure communication (TLS)

  Data Encrypted (by default)

→ **CosmosDB Partitions:-**

- Data items in cosmos are divided to partitions which are logical groups of items that are based on a specific property.

  ex:-

  In cars database, the Model can be a partition property.

Partition #1	Partition #2	-----	Partition #n
model: Mercedes	model Alfa Remo		Model: Toyota.

- Partitions are the basic scale unit in cosmos DB.

  ie; when we autoscale up (or) down, cosmos scalled is partitions.

  so, we add partitions

  remove partitions

- Make sure items are divided as evenly as possible.
- It's extremely important to select the right partition property.
- cannot be modified.

⇒ SQL vs NoSQL:-

Datastore
1. SQL
2. NOSQL

SQL:-
- Store data in tables
- Tables have concrete set of columns
- Query using SQL
- Very mature

NoSQL:-
- The greatest strength is scale & performance.
- Schema-Less
- data usually stored in JSON format.
- No standard for accessing data in NoSQL database.
  ↳ can be frustating.

Note:-
data is huge → NoSQL (un(or)semi structured)
data is Not huge → SQL (structured)

→ Today we have great JSON quering capabilities in databases such as "SQL server" and "postgre".

⇒ Cosmos DB consistency Level:-

Traditionally;

→ Relational DB - Strong consistenty : call returns only after successful comit in all replicas (-high availability)

→ NoSQL DB - Eventual consistency : call returns immediately commit in replica happens later (Low latency)

High availability ≠ Low Latency (performance)

→ cosmos offer five consistency levels:

1. Strong (≥ AS in regular relational DB)
2. Bounded Stanless
3. Session
4. consistent prefix
5. Eventual (≥ AS in regular NoSQL DB)

ex:-

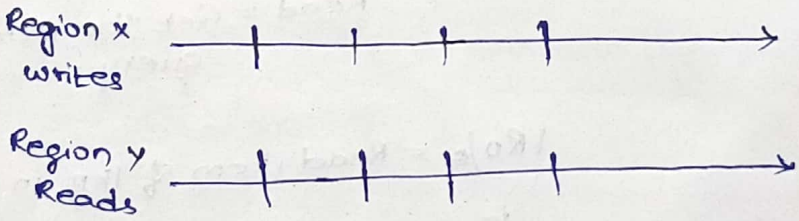
If region x updates an item, and region y reads this item which version will it get?

→ Strong:-

Region y will get the last version of the item updated in region x

- used for critical data.

Timeline:-



→ Stanless:-

Region y will be behind region x by K versions of T time ie;

gap will be never more than "2" versions.

- keeps the order of the versions.
- used for low write latency and when order is important.

→ Session:-

- In a client session - strong consistency
- other clients - use consistency prefix

→ Consistent prefix :-

- keeps the order of the versions.
- No gurantee of the lag size
- used for low write latency and when reads are infrequent.

→ Eventual :-

- No order garantee.
- No garantee of Lag size
- used for count of retweets, likes, ---etc.

⇒ Cosmos DB pricing :-

Based on RU's (request unit per second)

ie;

1RU = Read item of size 1KB

Here;

Read = Get the item by ID not by query.

1Ru/s = Read item of 1KB in 1Sec

ie;

400 Ru/s = Read 400 items of 1KB in 1 Sec.

Note :-

Update, delete, insert, query - more than 1RU.

You can see the actual RU consumed in the response header of the result.

→ Pricing also based on :-

- operation type
  1. Provisioned
  2. Autoscale
  3. serverless
- write regions
- No of provisioned RU's

→ Database operations:-

1. provisioned → predefined no. of Ru's can be changed. mutually later.
   offers reserved capacity upto 65% discount.

2. Autoscale → set the maximum Ru/s, cosmos scales upto this number.
   Good for unpredictable load.

3. Serverless → pay for what you use, No SLA

Handson:-

→ Creating CosmosDB:-

Azure cosmosdb

Click "create"

Select the "API" (Azure cosmosdb for NOSQL)

Basic:-

rg : readapp-rg.

account name : readit-cosmos (unique)

capacity mode : provisioned

Apply the free tier discount : Apply

Limit total - - : ☑ → By this is to avoid unexpected charges if we exceed the free allocation of 1,000 Ru/s.

Networking:-

connectivity : All networks

Minimum transport layer security : TLS 1.2

Backup policy:-

Backup policy : periodic

interval : 240

retention : 8

Storage

redundancy : Geo-redundant backup storage

→ Then "create".

→ Goto resource

Note:- Now we created an account, not a database.

If we want to creating data inside this account, then we need to create a database and in it a container of the data.

So;

Goto "Data Explorer"

Click on "New container"

ie;

database id : create new

name → readit-orders

throughput : manually

Ru/s : 400

container id : orders

indexing : automatic

Partition key : /priority

Click "ok"

Now, you can see the database in "DATA".

ie;

readit-orders

↳ orders

↳ items

↳ create new items

*for Querying :-

click on "sql button".

ie;

1. Select * from orders o
   ↳ alias.

2. Select * from orders o
   where o.total > 90

3. Select o.orderId from orders o
   where o.total > 90

4. Select * from orders o
   where exists (
   select value n
   from n in o.items
   where n.name = "Rama II"
)

↳ we can query the hirarchy of the document.

Note :-

we can edit any feilds or) values of the items but not the partition key.

→ Goto "keys"

Here we can see the keys that we are used to access and work with cosmos DB.

When we want to work with the cosmos db, we need to pass in a connection string.

ie;

this connection string comprisied of "URL" of database and the "key".

→ Connecting the orders function to cosmos DB :-

open the "orders" project in VS code.

Goto "processOrdercosmos.cs"

↳ uncomment line -19 & 26

Note:- Names in this code should match the cosmosdb in Azure, created.

and set the "connection string", in "Local.settings.json"

↳ "cosmosdb connection".

→ Run the code in local to test then deploy it to Azure function App

Azure MySQL :-

Managed MySQL on Azure.

works like any other mysql database using the same tools.

Great compalability with on-prem MySQL database

offers built-in security, backups, availability and more

Security :-

IP firewalls

Service endpoints

Pricing :-

Basend on :

1. tier
   - Basic – require light compute and I/O performance ie; dev
   - General purpose – Most business workloads
   - Memory optimised – require in-memory performance.
2. compute (no of vcores)

→ Creating and using Azure MySQL :-

Search "MySQL" (Azure database for MySQL servers)

Click "Create"

Then select type of database.

Select "flexible server"

Basic :-

~~rg : readit-rg~~

rg : mysql-rg.

server name : harish testmysql (unique)

region : East US

version : 5.7

workload type : for development or hobby projects

compute + storage : configure server. (nothing to change)

Authentication method : mysql Authentication

Networking:-

Add the firewall rule to allow our computer to access this.

→ "create"

After creating the resource.

Goto resource

↳ Goto "connect"

Here, we can see how to connect from various places.

Now, we use "Mysql Workbench",

→ Download "Mysql workbench" in local computer.

install "client only" while installing.

After installing,

Add the connection.

when you open the "mysql workbench" in connect from the Azure mysql.

you can see the steps to add connection to the workbench in your local.

So, Now you create Mysql Server.

Now create a New Schema (database)

⇒ Azure PostgreSQL :-

- Managed postgreSQL on Azure
- This version includes also Hyperscale deployment, which provides improved performance.
- offers built-in security, backups, availability and more.
- security
- Backup :-
  - → depends on storage size
    - upto 4 GB
      - full backup : once a week
      - differential Backup : twice a day.
      - Transaction log backup : every 5 min
- Retention period
  - 7-35 days (default is 7)
  - No Native long term retention support
- Availability
  - Backup is stored in a geo-redundant storage
  - SLA : 99.99%

⇒ Azure storage :-

- It is an "object store", a specialised kind of database that is used to store objects such as files, documents, vedios, images and so on.
- You will not use object store to store relational data, also you won't use it usually to store JSON documents.
- Masivelly scalable
- Accessible via HTTP and HTTPS
- durable and highly available.

Types :-

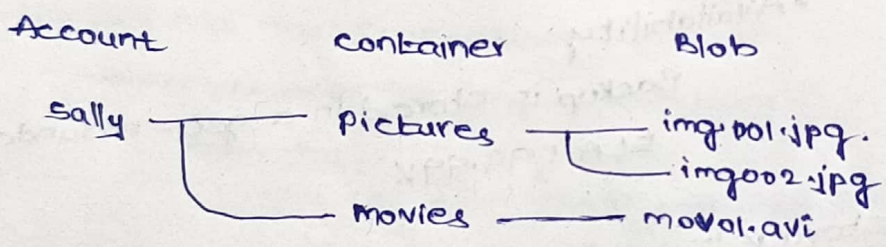
1. Blobs — object storage
2. Files — file shares for bound and on-prem deployments
3. Queues — queues
4. tables — NoSQL data store
5. Disks — storage volumes for Azure VMs

Azure Blobs Storage:-

- Blob — Binary Large object
- Great for files, videos, documents, large texts etc.
- you can upload upto 4.77TB per file (190TB per file in Preview)
- Extremly cost effective
- Masively scalable
- Extremely easy to use
  - ↳ there is a very simple API that allows to Azure storage

→ Azure storage is usually used conjunction with SQL or) NoSQL database. so that the database store the actual data of the system and blob storage stores the related files such as scanned documents, videos, images etc.

Structure:-



Redundancy:-

6 options

1. LRS
2. ZRS
3. GRS
4. GZRS
5. RA-GRS
6. RA-GZRS

Storage tiers:-

Blobs can be uploded to on of three tiers.

1. Hot:-

   data that's accessed frequently

Best SLA (99.9%)
Highest storage costs
Lowest access costs
ex:-
photos to display
documents to show.

2. cool:-
data that's access infrequently
Slightly lower SLA (99%)
lower storage costs
Higher access costs
Must be stored for atleast 30 days (or early deletion fees applied)
ex:-
short term backup
data for future processing.

3. Archive:-
data for archival
Stored offline, No SLA
can take hours to retrive
Lowest storage cost
Highest access cost
Must be stored for atleast 180 days (early deletion fee applies)
ex:-
long term backups

Note:-
retrival time is same in "Hot" and "Cold" tiers.

→ Creating a storage account:-

search "storage account"

create "New storage account".

Basic:- ~~rg : resource-app~~

rg : storage-rg

name : storageharish (unique)

performance : standard

Redundancy : GRS

Advanced:-

min TLS version : version 1.2

Networking:-

~~Public~~ Network access : public

data protection:- Enable soft delete.

↳ means if you delete any you undelete it for 7 days (default)

→ Goto resource

create a "container"

Goto "containers"

↳ add New one.

ie;

Name : storeimages

~~Private~~

Anonymous

access Level : container

To change the access Level to public.

Goto configuration

↳ Allow blob anonymous access - enabled.

Goto created "storeimages" container.

↳ upload

↳ upload any file.

in advance:-

type : blob blob

block size : 4mB

access tier : Hot

"upload"

→ To access the uploaded image, click on the image and then copy the "URL".

Note :- we can change the "access level"

overview
↳ change access level.

⇒ Accessing private blobs with keys and SAS Token :-

There are three methods to access the private blobs.

1. Azure AD identity.
2. Access keys.
3. Shared access signature.

Access keys :-

Goto Menu
↳ Access keys
Here we can see two keys & key & connection string
ie; while using client libraries to access storag account, then the librabries will ask the connection string.

Goto "shared access signature"
from here we can give the shared access.
by generating the "SAS token"

→ Creating storage Account for ReaditApp:-

rg: readitapp

name: orders-readit (unique)

performance: standard

"Create"

Now; create a New container.

↳ Add

Name: New-orders

Anonymus

access level: private

"~~Add~~" "Create"

This container will be used to store the new orders received by the app.

⇒ Azure Storage Explorer:-

It is a desktop app allowing you to fully managed storage account.

So, by this you can view your containers and your upload, download and delete blobs in the container.

Download "storage explorer".

↳ Install

Then "sigin to Azure".

↳ Then go to explorer.

So, from this you can do all thing same as on Azure storage account.

→ Azure Redis:-

Managed Redis on Azure.

Redies provide lightning-fast-in-memory distributed cache.

It is great for short lived frequently accessed data.

ie; shopping cart, stock quotes ---etc.

Pricing based on:

Tier

Memory

→ Creating Redis and connecting the catalog:-

Turn on "catalog VM"

search "Redies" (Azure cache for redies)

create a new one.

Basic:- rg : readit-app-rg.

name : harishreaditredies (unique)

cache type : Basic c0

Networking:-

connecting method : public endpoint.

"Create"

Goto resource

Here you can see the "Host name" which is used to connect to the redies.

Now, Lets configure our catalog to use redis, we want to connect the shopping cart to redies, meaning when we add books to shopping card from the catalog then these books need to be stored in redis.

After that, we are going to create/configure the cart model to read the data from redis.

open "catalog project" in VS code.

Set the "redies" connecting string in appsettings.json.

So, goto redies resource in Azure.

Goto "Access keys"

↳ copy "primary key" and primary connection string

Note:-

copy only till password.

So,

connection string that placed in the code should be;

"primary key @ host name ? SSL=true"

open "Index.cshtml.cs":-

Uncomment the lines

→ check the locally and then publish the updated code the VM.

→ Connecting the shopping cart to Redis:-

open "cart" project in VS code.

appsetting.json:-

add "connection string of redis".

add "orderfunctionUrl"

add "BooksDB connection string".

index.cshtml.cs:-

uncomment line - 37

→ check locally.

→ Working with Containers:-

Install "Azure CLI" locally from browser.

check in "cmd" after installation.

ie; az

open "cart code" in VS code.

Now, we want this code to run in AKS.

So, for that first we need to build a docker image based on this code and publish this image into a "container registory" in the cloud.

→ And for working with docker in this code, first we need to install "docker extension" in the VS code.

ie; Docker

using this extension, we can manage the containers, the images and the registories.

Docker file:-

docker file defines how we are going to build our docker image.

→ create "Azure container registory" (ACR)

ie;

rg: readit-app

name: readitminiacr (unique)

sku: Basic

→ "create"

Goto resource

↳ Goto "Access keys"

↳ Enable "Admin user"

this will help VS code accessing the acr and pushing code into it.

So, Now we can use this to build and store our docker image.

So,

Goto "VS code"

Goto "Docker extension"

↳ under "registories" you have Azure

↳ select your subscription

then you can see your container registory one that created.

Now, Goto "Docker file" in your code

↳ right click on this file and select "build image in Azure".

then; Name the docker we need to create

ie; cart:Latest.

then, it asks to selet the subscription

then, registory

then, OS : Linux

→ If you see error.

ie; cbl failed at 25. credentials.

then Goto terminal

↳ az Login

↳ login to azure

Then, agin build the docker image in Azure.

once, it is done

Goto "container registory"

↳ repositories.

↳ you can see your cart project.

→ creating AKS cluster & attach to the ACR :-

Note:-

deploy the "cart code" to the App service

& do same as we did for the

inventory app.

Messaging in Azure :-

- It is most important aspect in every system design.
- It is widely used between services.
- Messaging services must be able to handle load, throughput and have latency.
- It is a core part of every Microservice architecture.
- Azure has 4 fully managed messaging services.
  1. Storage Queue.
  2. Service Bus
  3. Event grid
  4. Event hub

⇒ Storage Queue :- (queuing service)

- It is part of Azure storage account.
- The simplest queue implementation.
- It basically have three components.

  create queue → send message into this queue

  receive message ↩

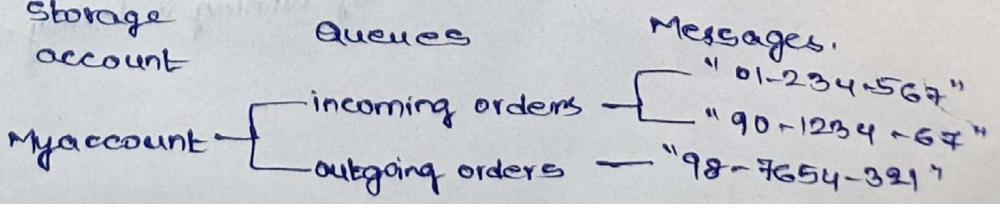
- No special pricing for queue, included in storage account.
- Same for availability.
- Performance :-

  for request of "1kb msg" size we have support of 20k messages per second per account (or) 2000 messages per sec per single queue.

- Note :-

  Max. msg size is ~~8KKB~~ "64KB"

- Architecture :-



→ Using Storage Queue :-

Goto "storage account"

Create "New"

rg : queue-rg.

name : storageQueueharish (unique)

performance : standard

redundancy : LRS

then, Goto resource

Now, Create New Queue

Goto "Queues" in data storage section

Create New Queue.

name : storeMessages

Goto the created queue.

↳ Add Message.

Enter the message

ie, {id:3, status:"success"} Expire in : 7

uncheck the box

"Encode the message body in Base64".

"ok"

Now, the message is waiting in the Queue.

so, we need to pull this message from here.

→ download the "StorageQueueDemo.Zip" and open this in VS code.

(this code will pull the messages from the storage queue and delete from the queue.)

In programs :-

add "connection String" (from access keys) and "queue-Name". (which is created)

Now, check by running it locally.

Event Grid :- (pub/sub service)

- Event Grid allows building event-based architectures, and that means that it publishes events to intrested parties.
- with Event Grid, there no queue and there is no order. So, there is no guarantee whatever that the order of events that are sent through the event Grid will reach its subscribers in the same order.
- It has strong integration with many Azure services.
- cost effective, simple pricing.
- No tiers, high availability in built-in.

Terminology :-

Event → It defines what happened. ex:- an event can be a storage blob was added or an IoT telemtery was received.

Publisher → who created the event.
ex:- Microsoft, my organisation.

Event source → where the event happened
ex:- storage account, IoT hub.

Topic → where the event is send

Subscription → which events intrest me.

Event handler → where the event is sent.
ex:- I want the events to be sent to Azure func (or) Event hub (or) to my custom code.

- SLA - 99.99%
- Max-Event size : 1MB
- performance - supports 10M/sec (or) 5k events/sec/topic.

Pricing :-

Based on :

1. Number of operations
   first 100k operations are free.

→ Connecting Event Grid to the Order function:-

Run the Catalog VM

Search "Event Grid" in Azure

ie; (Event Grid system topics)

↓

System topic in event grid is basically triggred on event that are built-into the system.

So, the event that triggered when new blob is created in a storage account.

↳ Create New one.

ie;

Topic type : storage account

↳ because the event that we want to trigger is a result of a blob that is added to storage account.

rg: readit-app-rg

resource: ordersreadit

which we want to base the topic-on.

ie; storage account that contains the orders.

Name; ordersbopic

→ "Create"

ie; when a new blob (or) a new order will be stored in a storage account then a new event will be triggered and handled by the event Grid.

&

event Grid trigger the orders function instead of the http trigger, thus connecting the storage account to the function.

Now, we need to do some modification to the function app, because our function is currently triggered by the http (or) a rest-API port but we want is that to trigger this function as a result of an event of the event Grid.

download the "processorderCosmos.zip" and replace the file in the order project.

Now, open the "order project" in vs code.

open terminal.

↳ dotnet add package Microsoft.Azure.webjobs.Extensions.EventGrid.

(to download the Event grid package)

In local.setting.json:-

add the "storageconnectionstring".

and add new line;

"AzureWebJobsStorage": here we need to place the connection string of the storage acc used by the "function app".

ie;

readit-funcstorageminu.

Note: we don't have the storage account for the function app. so, we will create the new storage acc & add its connection string to the function app.

ie; Goto func app
↳ configuration
↳ Add New application setting.
ie; Name = AzureWebJobsStorage
Value = connection string of storage account.

→ deploy the code to "function App"

↳ click "upload settings"

Now, we need to define the event Grid to trigger this function.

Goto "order-topic"

↳ "Add event Subscription"

ie;

Name : order-subscription.

Event types : Blob created

Endpoint type : Azure function.

Endpoint : select the "function" in function app.

→ "create"

Now;

Goto "Event Subscription"

↳ you can see the created one.

Test :- storage account → orderreadditems.
function
cosmosdb

Now, upload a file or an order to the container and we like to see this order being read by the function and then store inside the cosmosdb.

→ upload "ordersample.json" from orderproject to the container in storage account.

→ Now, check in the "processordercosmos" wheather there are logs.

→ Now, check in the cosmosdb wheather the order is added to this.

→ Connecting shopping Cart to the Storage account :-

replace the "index.cshtml.cs" file in the "cart project".

In appsetting.json :-

add connection string.

↳ "StorageConnectionString" : ↗ container one Not the function app storage

replace this with "orderfunctionUrl"

→ deploy the code.

→ protecting the orders function :-

making only to be accessible for event grid.

Goto "orders function"

↳ Networking

↳ "Access restriction"

Now;

Add add New rule

ie;

Name : allowEventgrid

Priority : 100

type : service tag.

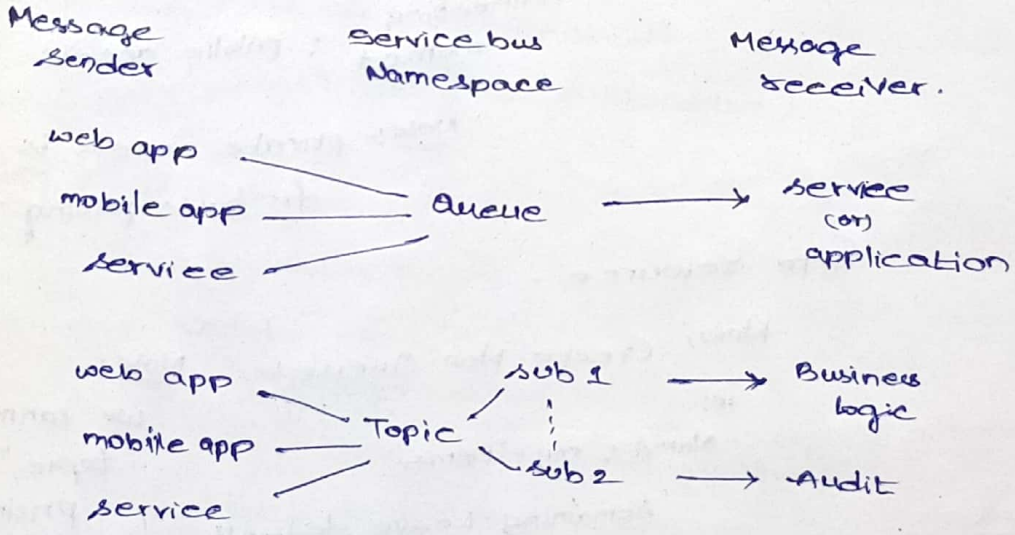
service ~~tg~~tag : AzureEventgrid.

"Add rule".

⇒ Service Bus :-

- full managed, full-blown message queuing service built into Azure.
- durable
- supports point-to-point (queue) and pub/sub (Topic) seenarios.

service bus queues :-



Advanced features :-

↗ first in first out

1. Message sessions (FIFO)
2. dead-letter queue
3. Scheduled delivery
4. transactions
5. duplicate ~~deletion~~ detection

- SLA – 99.99%
- can be configured for geo-disaster recovery
- Security.
    1. IP Firewalls
    2. service endpoints
    3. private endpoints

→ Using Service Bus :-

Search "Service Bus"

Create

ie;

rg : sb-rg.

Namespace name : sbdemomemi (unique)

pricing tier : Basic

Networking :-

connecting method : Public access

Note:- private access is not available for basic pricing tier.

Goto resource.

Now, Create New Queue.

ie;

Name : my-items

remaining Leave default

"create"

Note:-
we cannot create "topic" in basic pricing tier.

→ download the code.

it sends the message to the service bus.

↳ add connection string.

Goto "Access policy" for connection string in service bus

Then "run" the code.

for checking the Sent Messages.

Goto "Service Bus Explorer"

↳ click on "peek from start"

↳ click on the message for the body.

⇒ Event Hubs :-

- Big data streaming platform.

Note:-
No messaging" in the description.

- It is basically a managed kafka implementation

↳ kafka defined as big data streaming platform not as a messaging platform.

→ Most clients use this.

- can receive millions of events per second.
- SLA - 99.99%. (dedicated)
  99.95% (basic & standard)

→ Using Event Hubs:-

Search "Event hubs"

Create New one

ie; rg : hub-rg

name : endemomini (unique)

Pricing tier : basic

Networking :-

connecting : public method.

Goto resource

Now, create "Event hub"

ie;

Name : my-telemetry

"Create"

→ download the code.

it create the consumer group for the event hub and then create consumer in the group and then while true it listnes to events from the event hub and every message that is received is written to the console.

Set the "connection string" of the event hub.

↳ from "shared access policies"

↳ click on policy

↳ copy the conn. string primary key.

"Run" the code.

Now, Goto portal

↳ Goto "Generate data"

↳ by this we can stream data to the event hub and we stream various kinds of data.

Now, you can check in the console of the vscode. you will receive the message.

ie; pulling messages from the event hub.

⇒ Selecting Messaging Solution :-

Service	used for	Guarantee order	Max Msg size
Storage Queue	dead simple queeing	Yes	64KB
Event Grid	Event driven architecture	No	1MB = 1000KB
Service bus	Advanced queueing solutions	yes	256KB
Event hubs	big data streaming	yes	1MB

Identity Managment with Azure AD:-

Microsoft changed the name of "Azure AD" the identity managment service in Azure to "Microsoft Entra".

Azure AD:-

- Azure active directory. (external resource)
- It is a central identity and access managment cloud service and it is used to manage access to thousands of apps.
  - ↳ one of them is "Azure".
- Secure, robust, intelligent.
- It has MFA, conditional access, device managment hybrid identity, monitoring reports --etc.

→ we mainly intrested in;

controlling access to the resources.

↳ by setting up users, groups, roles

And, we use Azure AD to add "Authentication to our apps. (readit-app)

We can also done via Azure AD B2C.

Tenant:-

It is a special instance of Azure AD containing accounts and groups.

It is also called as "Directory".

It is not part of the subscription hierarchy.

↳ exists besides the subscription.

for new subscriptions, a new tenant is created automatically.

Note:-

A tenant can be assigned to multiple subscriptions.

→ Look around Azure AD:-

Search "entra" in portal

open "Microsoft Entra ID"

we can see, Name as "default directory".

to change that;

Goto "properties".

here you can change the name.

ie; AzureCourseDirectory

→ Goto "users"

↳ here you can see "Azurecourse Student".

Goto the "User"

Here, we can see;

object ID (we use later)

assigned roles

so, you can see the roles.

ie; "Global Administrator"

→ Goto "sign-in logs"

Here, we can see the recent sign-in's that we did to this tenant.

→ Goto "password reset"

Here we can configure self-service password reset, meaning the users that are registered inside the tenent can reset their own password.

Note:- this feature is not available for "free tier".

⇒ Users and groups:-

Two of the main three objects managed by Azure AD.

↳ third one is "roles"

- Manages and stores the users that are part of the tenant
- Groups the users in Groups

ex:- IT Admins, developers -- etc.

- using groups, it allows defining roles to groups instead of each user.

→ Working with users and Groups:-

Goto "Microsoft Entra ID"

Goto "Users" and create New user.

ie;

user principal name : david

display name : david jones

copy the password which Azure is created.

Assignments:-

here we can assign groups and roles here.

Note:-

copy the "user principal name" & "password".

and save them somewhere.

Now,

Goto Incognito mode;

signin as "@davidjones"

↳ It asks to update the password.

Note:-

david jones has no subscription.

↳ we did not authorize this user for a specific Azure subscription.

So, for that add "roles" to david jones.

→ "Also we can invite external user" for the tenant.

↳ ie; user already has email id.

ie; Goto "All users"

↳ Add users

↳ Add "invite external user"

ie;

Email : mimi@example.com

display name : Memi Lavi

Message : Hi there, You're invited to my Azure AD tenant :)

Now, create "Groups"

Add "new group"

ie; group type : Security

group name : Azure Architect.

members : add "david jones" user for this group.

⇒ Azure AD licenses:-

Azure AD licenses have great effect on the functionality and price of Azure AD.

*** MFA:-

Multi-factor Authentication.

- Authentication is divided into three factors.

1. something you know → (username/password, security question)
2. something you have → (phone, smart card)
3. something you are → (fingerprint, iris scan, other biometric data)

Two factor Authentication:-

for extremely sensitive systems
ie; banks, -- etc.

ie; we pick two of the three factors.

ex:-

something you know + something you have & other combinations → (username/password + code in Text)

⇒ Azure AD security Defaults:-

we saw, full MFA protection, exists only with P1 and P2 licenses.

With security defaults we can increase the protection of the sign-ins even in the free tier.

→ Using security defaults:-

Goto "Entra"
↳ Goto "properties"
In this you will find "security defaults"

Now, click on "Manage security defaults"
↳ you can disable here.

⇒ Role Based Access Control (RBAC):-

In the past, authorization was defined per user (or) user group.

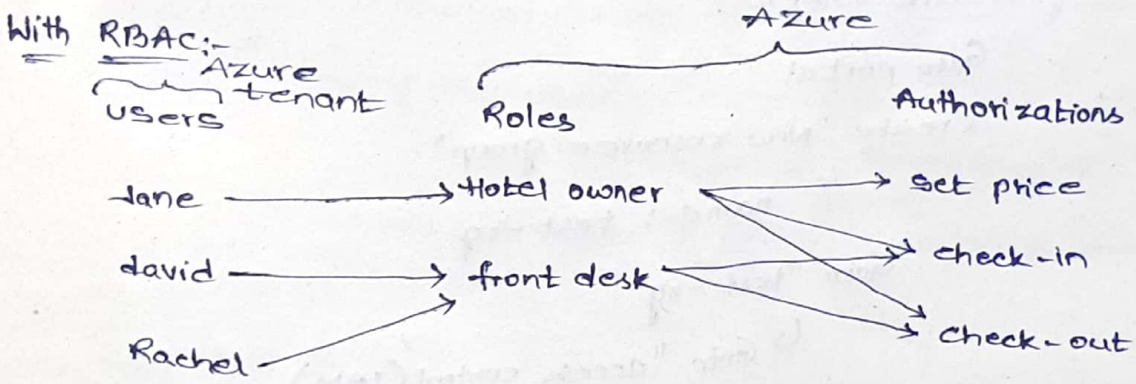
ex:-

david is allowed to add items to the inventory.

John is not allowed to read data of other doctors.

This is extremely had to maintain.

With RBAC:-



Azure roles:-

In order to perform any operation, or access any data in Azure you have to have the appropriate role.

ie, if you want to;

- create resource groups
- Access data in SQL
- see metrics of App service

:

then you have to have the right role.

There are three types of roles:-

1. Owner → can perform any action on the resource, including assigning roles to it.
2. contributor → can perform any action on the resource, but cannot assign roles to it.
3. reader → can only view data, but cannot change anything.

ex:-
we have three roles.

1. virtual Machine contributor → can manage VM
2. cosmos DB account reader → can read Azure cosmosdb account data.
3. service bus data owner → Allows full access to service bus resources.

Note:-
It always better to assign roles to groups and not individual users.

→ Using Azure roles:-

Goto portal

Create "New resource group"

ie;

name : test-rg.

Goto "test-rg"

↳ Goto "access control (IAM)

Here we can assign roles and manage roles.

Goto "role Assignments" tab.

click "Add" (Add role assignment)

Now add "contributor" role in Privilaged administrator roles to "Azure Architects".

Now,

signin as "Azure Architect"

↳ you can see "test-rg" in resource groups.

⇒ Managed Identities:-

The ability to assign Azure AD identity to Azure resource.

The resource can connect to other Azure resources using this identity.

No Need to handle credentails (username, password, etc)

→ Using Managed Identity with Inventory App service:-

Goto " inventory app-service".

Goto "Identity"

Enable "System assigned" managed identity

↳ for this, so that this app service will have a user which we can use later to access our Azure SQL instead of using username & password.

status : "on"

Now;

So, a new user was created in our identity system, which is Azure Entra (or) Azure AD.

&

name of the user is name of the app service

ie; "mini-readit-inventory".

So, we use this user in order to access our database.

→ Goto "SQL Servers"

Goto "Microsoft Entra ID" in settings section.

↳ set admin

↳ "select user.

ie; ~~Azure~~ AZcourseStudent

Now, we can query the data in "query editor" using a user in Microsoft Entra.

Now, run the queries in the query editor by replacing the app service name.

Then;

Goto "App services"

↳ Goto "configuration"

↳ edit the connection string.

remove the "username & password" part and add;

Authentication = Active Directory Managed Identity

So, now we are not using username and password.

→ test by running the inventory app service.

So, our App service is currently connected to the database using "Managed Identity" and Not using username and password.

This is much more secure than the traditional way of connecting to database and it is recomended way to do it.

⇒ Using Azure AD on our App:-

The process:-

- register the app in Azure AD
- Add code to use Azure AD as authentication engine.
  - ↳ for app service - can be configured via the portal.
    and there is no code changes required.

The authentication:-

uses OAuth and JWT

→ This is to protect the inventory app for unauthorised access by users who are not the store's employees.

⇒ OAuth 2 :-

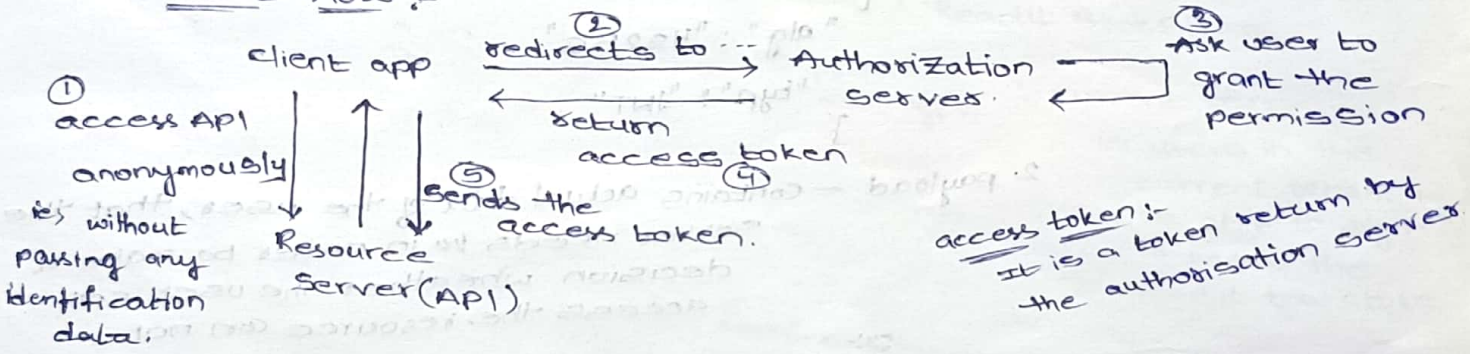
It is a standard protocol for authentication and authorization.

It is widely use, mainly in web apps.

Working :- (components)

User	The user who want to access protected resource in API
Client App	The client application accessing the API
Authorization server	Authorize the user for the client application.
Resource server	The API being accessed.

OAuth 2 flow :-



access token :-
It is a token return by the authorisation server.

Ex :-

We want to login to "feedly" site.

It asks we to login, by selecting the Authorization server.

ie; facebook
google ---etc.

Suppose, we select google
then we enter the login credentials.
So, then google Authorization server identify me and sends the token to the feedly that I known this guy.
ie; google identify me for feedly.

App registration :-

- Authorization server should be familiar with the resource server (API)
- Resource server must register itself with the Authorization server.
  ie; name, url, -- of the resource server.

Note:- After registaring.
resource server must pass the "client ID" and the "client secret" to the Authorization Server.

JWT:- (access token)
Json web token

- It is string containing data the server needs in order to authenticate the user.
- This token is return by the "OAuth2" Autherization server.
- JWT has three sections :-
  1. Header — type of token (JWT) and signing algorithm
     ie;
     ```
     {
       "alg": "HS256",
       "typ": "JWT"
     }
     ```
  2. Payload — Contains actual data of the user, that the server needs in order to make a decision wheather the user can access the resource (or) not.
     ex:-
     ```
     {
       "sub": "1234567890",
       "name": "John Doe",
       "admin": true
     }
     ```
  3. Signature

- three parts of JWT are "Base64 encoded" and concated with dot(.).

ie;
```
eyJhbGciOiJIUIN.        → header
eyJzdWzHiLMOILMOPqrAbce.  → payload
4P69chReOMZWxU4
                        → signature.
```

→ Configuring App service for authentication using Azure AD :-

Goto "inventory app service"

↳ allow direct access to this app service.

Now we can configure it for Authentication.

Goto "Authentication"

↳ "Add identity provider"

ie;

identity provider : select Any (Microsoft)

tenent type : workforce

app registration : create new app registration.

Name : Readit Bookstore.

Supported acc type : current tenant

ie; users in the current tenant will be able to sigin to access the readit bookstore.

Restrict access : Require Authentication

unauthenticated requests : Http 302 found redirect.

→ "Add"

So, now authentication is enabled for our App service.

Now,

Goto "Microsoft Entra"

↳ Goto "App registrations"

↳ Goto "All applications"

Here, we can find the app registration we just created.

→ Adapting the inventory code and using Azure AD:-

Now, we want to change the code, to show the logged in users.

download the "InventoryAzureADAuth.zip" file.

So,
replace the files & folder with the existing one's in the inventory app project folder.

open the code in VS code.

& add the packages that required. (from the file azuread-auth-command.txt)

open "appsettings.json":-

here we need to set the values.

So, Goto "portal"

↳ Goto "Authentication" in inventory app service

copy the "client ID"

Goto "Entra"

↳ copy "tenant ID"

→ Paste these in the appsettings.json file, to route to the correct tenant ID and the client ID.

Then, deploy the code to App service.

Now, you can see the "user name" in the page.

Note:-

After testing.

In the url, add

1. /claims → Here you can see all the claims that will received.

2. /.auth/me → Here, we can see the token that was received from Azure AD.

→ copy this "token" and go to "jwt.io" in browser and paste the token.

then, we see the various data that is transmitted.

Monitoring in Azure :-

A working app is not enough.

we must known its status, how it performs, and when it has problems

This is done via monitoring.

- Azure offers a lot of built-in monitoring mechanisms.
- There is a centralized monitoring hub where all monitoring data is streamed, and can be queried, or used for the triggers.
- It is extremely cost effective.
- Monitoring is based on two types of data.

1. Metrics → Numeric values describing resource's aspect at a specific point of time.
   Ex:- CPU, disk performance time.
2. Logs → Events that occured in the system
   textual & numeric
   Ex:- the log can document when the system has started, and when an error has occured and so on.

⇒ **Resource Monitoring :-**

Almost every resource in Azure has a "Monitoring" section.

⇒ **Using Metrics :-**

Goto "catalog VM"

Start the VM

Goto "Monitoring" section in overview page.

Here we can see the metrics.

We can see "particular Metrics"

ie, percentage cpu

Similarly, we can check for different resources.

Note :-

We can save the Metrics to the dashboard, and customise.

⇒ Alerts:-

Get notifications about events in your resources.

ex:-

Get alert when VM's cpu goes above 90%.

(or)

when more than 20% of request fails in app service (or) other app.

(or)

when server error occurs in the last hour.

Components:-

1. condition → when to trigger the alert
2. action → what to do? usually - send notifications
   Notifications are sent to action groups.
3. Details → contents of notification.

→ Using alerts:-

Goto "catalog VM"

Goto "Metrics"

Now, we are going to define alert for cpu utilization.

ie; select "percentage cpu"

expand time for 30 days.

Now, click on "New alert rule"

condition:-

configure the condition.

put "90%" (90) in threshold.

Actions:-

"create action group"

ie;

rg: readit-rg

group name: course-admin-rg.

name: course-admin.

Notifications:-

type: email/sms msg/push voice.

select "email"

give email.

enable the common alert schema

Name: email to admin

actions:-

type : (Not defining Now)

"Create"

→ Test action group

Select sample type : Metric alert static type.

Details:-

severity : 2- warning.

name : High cpu utilization

description : cpu of the VM is high. please take a look.

→ "create"

Now, alert rule was created.

Goto "Alerts"

↳ click "Alert rules"

⇒ Logs and Analytics Workspace:-

- Almost every Azure resources generate logs.
- logs records need central repository to be stored and viewed, this is log analytics workspace.

Log analytics workspace:-

A central location for storing, organising and analyzing logs.
Aggregates logs from all connected, monitored resources.
used specialized query language to query logs.

→ Creating & using log Analytics:-

Search "log analytics" (log analytics workspace)

→ Create New

rg: readit-rg

Name: logs-ws-memi (unique)

Goto resource.

Goto "Logs"

here, we can query & analyse logs in the system

Click on "select scope"

Select any one here.

ie; select our subscription.

Note:-

We need to explicitly configure the resource to stream its logs to our log analytics workspace.

So,

Goto "Subscription"

Goto "Activity logs"

"click on "Export to activity logs"

Add "diagnostic settings"

ie;

Name : acitivity-logs-aw

logs : Admistrative,
security,
recomandations.

destination details : send to log analytics workspace. ⇒

"save"

Now, Goto "log analytics workspace"

↳ change the query.

ie; AzureActivity & then run.

⇒ Insights:-

- A collection of metrics, statistics and insights about the resource
- specific to resource type
- Generated automatically.
- code base services (App service, apps and VM's) can integrate application insights into the code and gain a lot of data about app usage, performance, --etc.

→ Using insights:-

Goto "mimiorderreadit" Storage account.

Goto "Insights"

Here, we can see the data.

Now, Goto "catalog VM"

Goto "Insights"

for VM we need to "enable" some monitoring capabilities of a VM.

So, click on "enable"

↳ then "configure"

→ deployment will take 5-10 mins.

After, Goto "performance" tab.

↳ here we can see the performance of the VM

⇒ Azure Monitor:-

- It is a central location for all the monitoring aspects of Azure's resources.
- Provides access to metrics, logs, insights and more---
- and also has some additional capabilities which are not found in the individual resources.

→ Using Azure Monitor:-

first start the "catalog VM".

Search & open "Monitor"

→ we need not create a monitor, it is already created and pre-configured.

from here we can see the Metrices, logs, insights for different resources.

→ using Application Insights :-

Goto "Monitor"

Goto "applications"

→ we can also directly search "application insights".

↳ great tool to monitor and troubleshoot web apps.

Goto "readit-inventory"

Goto "Availability"

↳ Here we can define availability tests, which will continious monitor our web app and make sure that its alive and accessible.

So, "Add standard test"

ie,

test name : inventory-av-tests

URL : Put the default domain of the app service.
(Put "https://" before it)

test frequency : 5 min

test locations : select some (default)

click "create"

Test will be created under "availability test".

click on "..." of the test and go to "open rules(Alerts) page".

Here, a new alert is automatically created for us, this alert will warn us when a test was failed.

click on that "alert"

↳ Goto "Action Groups"

Here we can set the action group ie, what should be done when an alert is trigger

And, Now we can see the test results in the "Availability" page of "memi-readit-inventry" of the Application insights.

⇒ Tags :-

- Resources are organised into resource groups.
- Sometimes we want more information about resources. ie; which environment it is ?
  ie; Test (or) prod

  To which app it belong ?
  ↓

  so, for these we have "resource tags".
- Tags help organize resource using "name-value pair".

  ex :-

  environment : test
  App : readit
  Group : A-team
- Tags can be set during the creation of resource or after that.
- usefull for many resources :
  1. resource quering ie; show all resources of A team.
  2. cost analysis ie; how much did the test environment cost last month ?
     ↓

Security in Azure :-

⇒ Key Vault :-

- Many apps have secrets that need to be kept safely.
  ie; connection strings,
  keys for encryption & decryption.
  certificates
  API keys ---
- usually these secrets are kept in the configuration files, configuration DB etc.
  ↳ (Not really secure)
  so "key vault" solves this problem.
- "key vault" safely stores secrets of various types.
- It has very restricted access, it needs Azure AD authentication in order to access the secrets and other confidential information that is stored inside it
- Easily manageable as we will see, and is accessed via "REST API".
- It is cost effective.

→ using keyvault in catalog VM :-

Goto "key vaults"
create New one.
rg: readit
Name: harish readit keyvault
Price
tier : Standard.

access configuration :-

Permission model : Azure role based access control mode

Now,
configure "catalog code" to work with key vault.

Goto program.cs:-

we are going to add support for key vault so that the connection string will be read from key vault instead from the configuration file.

→ first "add package" to code to support key vault.
ie;" dotnet add package Microsoft.extensions.configuration.AzureKeyVault".

→ download the "program.cs" file from lecture resources, and replace the file.

Goto "portal"

↳ Goto created "keyvault"

Here, we can store keys, secrets, certificates - - -

→ Let's add diagnostics settings to this key vault so, that we trace the activity.

Goto "Diagnostic settings"

"Add"

Name: kv-settings-all

category type : audit

Metrics : AllMetrics

destination details : send to log Analytics workspace.

↳ select the one we created

→ Let's add the connection string secret to this key vault.

Goto "Secrets"

→ we will see a message that, currently we are not authorized to view & make changes to secrets.

So, we need to add role for allowing us to manage secrets.

Goto "Access control (IAM)

click "Add" (Add role assignment)

↳ select "key-vault secret officer" role

Next

↳ select member.

"Add role"

Now, Go back to secrets.

click "Generate/imports"

Name : ConnectionStrings--BooksDB

secret

Value : copy the connection string from the app.settings file of the book db and paste it here.

"Create"

Note:-

since we added the Azure key vault configuration provider to our code that means this configuration provider is going to look for a very specifically named secret in key-vault.

Now, go to "overview" page

↳ copy Url of vault (ie; Vault Url)

Goto "code" and remove the connection string of books-db and place this "vault url".

ie;

```
"KeyVault" : {
    "BaseUrl" : "vault url value".
}
```

Note:-

We cannot run this code locally, because local code doesn't have permission for key vault.

So, publish the code to VM.

Note:-

We might see some errors.

Reason:-

we didn't allow the VM to access the key vault.

So, allow the access to VM in key vault.

Goto "key vault"

Goto "access configuration"

We can "check" the Azure VM for deployment option in resource access.

(or)

assign a role to VM to allow the access.

for that,

first we need to define identity to VM

so,

Goto "catalog VM"

Goto "Identity"

"Enable it"

Goto "keyVault"

IAM

"Add" (Add role assignment)

↳ select "key vault reader" ⊗

"key vault secrets user" ✓ role

and in "members"

Assign access to: Managed identity

members: select VM

↳ select "catalog VM"

"Assign"

Now, role was assigned.

So, now VM can connect to the key vault.

DR in Azure :-

DR - disaster Recovery.

- It is a plan to recover from a complete shutdown of a region.
  - ↳ usually as a result of a disaster (earthquake, flood, -- etc)
- Some apps require it and some don't.

Note :-

A complete shutdown of a region is extremely rare.

→ How DR works?

Inorder to setup DR, we need to do the following?

- Select a DR site.
  - ↳ A secondary region that will function as ou primary in case of a disaster.
- configure it to be ready for activation when necessary.

Hot / cold :-

Hot DR → when the primary region went down, the secondary region will automatically active and all the user immediately began working.
- Nok data loss
- Requires duplicate infrastructure
- It is most expensive method.

Cold DR →
- It takes sometime for active when primary went down.
- Might be manual
- some data might be lost
- Less expense.

RPO / RTO :-

RPO →
- Recovery point objective
- How much data we allowed ourselves to loose in case of disaster.
- usually measured in minutes

ex:- we have an RPO of 5min. ie; we lost last 5min of data.

RTO →
- Recovery time object
- How long it should take before the system is up agai
- measured in minutes.

⇒ Routing in DR :-

During DR users should be routed to the secondary region.

three Methods :-

1. Inform the user about the new address of the app.
   ↳ for this case, there isn't any routing chain, because the users will simply direct their browsers to a new app and we don't have any central routing mechanism such as "DNS".
2. Manually change DNS record to point to the secondary region.
3. use automatic routing.

→ Azure has "2" automatic routing services.

1. Traffic Manager
2. front door

Azure Traffic Manager :-

It is a DNS based load balancer.

Enables traffic distribution across global regions.

Working :-

A user makes a DNS query to the traffic Manager profile, and the DNS name is like "www.abc.com" then traffic manager decides which region to go to, and this decision is based on various algorithims and then the traffic manager return the IP address of the appropriate region to the user.

→ Using Traffic Manager :-

Goto " Traffic Manager profiles"

Create New one

Name : harishreadit (unique)

Routing method : priority

Rg : readit-app-rg

Goto resource.

Goto "configuration"

Probing interval : 10 sec → How frequent traffic Manager should check the health of the end points.

Tolerant no. of failures : 1 → After how many failures it should switch the region. (endpoint)

Probe timeout : 9

Now, we want to demonstrate how to use traffic Manager for routing b/w endpoints.

for that we are going with the inventory app, because the inventory service using the standard port as opposed for example for catalog which uses non-standard port and this makes the configuration more complicated.

So,

Goto "Endpoints"

"Add"

Type : Azure Endpoint

Name : readit-primary

target

resource type : App service

Target resource : inventory-app

Priority : 1 (means whenever it is available, the traffic manager should route to this endpoint)

Health checks : enable

Note:-

disable the authentication for inventory app service.

Now, check;

copy the DNS Name of traffic Manager and browse.

if you see error ie, your connection is not private.

then; Advanced

↳ proceed to - - -

Now, create another app service

rg : readit

Name : harish-inventory-dr

Region : other than you normally using.

So, Goto "Traffic Manager" and configure the 2nd endpoint.

Goto "Endpoints"

"Add"

type : Azure endpoint

Name : readit-secondary

target : App service

target resource : inventory-dr

priority : 2

Health check : enabled

→ Check wheather traffic Manager is routing the traffic by stoping the primary endpoint ie, inventory app.

⇒ Azure front door :-

- Global Entrypoint for web apps
- works on layer 7 (Http/Https)
- multiple routing methods
- Similar to application gateway but in global scale.
- URL path based routing.
- session affinity
- SSL offering
- WAF integration
- URL rewrites
- Http/2 support

Using Azure front DDOR :-

Goto "front Door and CDN profiles"

Create New one

Select "explore other offerings"

↳ Azure front door (classic)

basic:-

rg : readit-rg

configurations:-

Here we are going to configure the front end, the backend and routing rules.

frontend:-

Host name : readit-harish (unique)

"Add"

Backend pools:-

Name : readit-pool

Add a backend

↳ add the two app services.

ie; inventory

inventory-d

ie; host type : App service.

name : inventory

Http port : 80

Https port : 443

priority : 1

weight : 50.

Suppose, if we have two or more backends with the same priority then the traffic will split b/w them based on weight.

Similarly Add 2nd backend.

ie; Priority : 2

And, change

Prod interval : 10

Sample size : 1

Successfull samples : 1

routing rules:-

Name : readit-rule

"Add"

→ "create"

Now, Goto resource (front door)

we can check using "frontend host URL"

Note:-

We can add access restriction rule, to allow traffic only through to the front door.

ie; Name : frontdoor-access
type : service tag

service tag : AzureFrontDoor.Backend

⇒ Traffic Manager Vs front Door:-

Generally, if you need Http related capabilities go with front door.

ex:-

url-path based routing.
SSL offloading
web application firewall

Otherwise, Go with traffic Manager — usually cheaper.

Note:-

front door might respond much faster to changes in the backend pools.

Traffic Manager, might take time.